Títol: Audioteca de RAC1

Autor: Sinuhé Goñalons

Data: Setembre 2016

Director: Omar Isaac Capó Escrivà
Institució del director: Grup Godó de Comunicació, S.A.

Ponent: Carles Farré Tost
Departament del ponent: Enginyeria de Serveis i Sistemes d'Informació

Empresa: Grup Godó de Comunicació, S.A.

Titulació: Enginyeria Informàtica (pla 2003)
Centre: Facultat d'Informàtica de Barcelona (FIB)
Universitat: Universitat Politècnica de Catalunya (UPC) BarcelonaTech

Membres del tribunal

Secretari (ponent): Carles Farré Tost
Presidenta: Claudia Patricia Ayala Martínez
Vocal: Rodrigo Ignacio Silveira

Índex

1 Introducció **4**
- 1.1 L'empresa . . . 4
- 1.2 Antecedents . . . 4
- 1.3 Objectius . . . 6

2 Especificació **8**
- 2.1 Anàlisi de requisits . . . 8
- 2.2 Diagrama de seqüència . . . 11

3 Disseny **13**
- 3.1 Arquitectura física . . . 13
- 3.2 Arquitectura lògica . . . 18
- 3.3 Model de dades . . . 20

4 Implementació **23**
- 4.1 Entorn de desenvolupament . . . 23
  - 4.1.1 Repositori de codi . . . 23
  - 4.1.2 Maven . . . 23
  - 4.1.3 Jenkins . . . 23
  - 4.1.4 IDE . . . 23
- 4.2 Principals tecnologies . . . 24
  - 4.2.1 Java . . . 24
  - 4.2.2 Spring . . . 25
  - 4.2.3 Lucene . . . 26
  - 4.2.4 Edge Side Includes (ESI) . . . 26
  - 4.2.5 Project Lombok . . . 27
  - 4.2.6 Velocity . . . 28
  - 4.2.7 ZK . . . 28
  - 4.2.8 Mp3agic . . . 28
  - 4.2.9 Validator . . . 29
  - 4.2.10 jUnit . . . 29
- 4.3 Parts destacades . . . 31
  - 4.3.1 Sincronització Lucene . . . 31
  - 4.3.2 Dades persistides . . . 33
  - 4.3.3 Servidor d'àudios amb publicitat . . . 34
  - 4.3.4 Migració . . . 39
- 4.4 Instal·lació i manteniment . . . 40

5 Planificació i valoració econòmica **42**
- 5.1 Calendari . . . 42
- 5.2 Cost . . . 45

6 Conclusions **46**
6.1 Millores proposades 46
6.1.1 AdsWizz 46
6.1.2 Migració 46
6.1.3 Player 46
6.1.4 Backup 46
6.2 Valoració del projecte 47

7 Annex. Imatges de l'aplicació **48**

Índex de figures

1 Web antiga: Portada 5
2 Web antiga: Pàgina de programa 5
3 Web antiga: A la carta 6
4 Diagrama de seqüència. Upload/update d'un àudio 12
5 Diagrama de comunicació 14
6 Méthode Client. Cerca d'àudios 16
7 Méthode Client. Disseny de la portada 16
8 Agent de gravació 17
9 App (mobile iOS) 19
10 Disseny en capes 19
11 Entitats principals 20
12 Edge Side Includes 27
13 Sincronització de Lucene 32
14 Servei d'àudios amb publicitat 38
15 Calendari de desenvolupament proposat (1/2) 43
16 Calendari de desenvolupament proposat (2/2) 44
17 Home de la nova web. L'audioteca mostra el que està sonant ara i el llistat de players 48
18 Detall d'un player en una notícia 48
19 A la carta. Resultats de cerca i playes estàtic 49
20 Detall de graella de programes 49
21 Notícia per a compartir a Facebook i Twitter 50
22 Backoffice. Gestió d'àudios 50
23 Backoffice. Gestió de programes 51
24 Backoffice. Gestió de graella 51

1 Introducció

1.1 L'empresa

Grup Godó és un *holding* de comunicació. Va ser constituït el 1998, però els seus principals mitjans de comunicació van ser creats molt abans. Els més importants són La Vanguardia, diari generalista creat el 1881, i Mundo Deportivo, del 1906, especialitzat en premsa esportiva. Tot i que el principal mercat es troba a Catalunya, els dos es distribueixen a tota Espanya i fins i tot tenen diferent edicions locals. Cadascun d'ells també té la versió web.

A més d'aquests dos diaris, els mitjans inclouen ràdios (RAC 1 i RAC 105), televisió (8tv), revistes (Què Fem?, Vanguardia Dossier, Magazine) i webs (GurusBlog). Entre la resta de negocis del grup es troben una empresa dedicada a gestionar la publicitat (Publipress), una planta d'impressió (CRE-A) i una empresa de distribució (Marina BCN Distribucions).

1.2 Antecedents

El projecte més important tècnicament dels últims anys ha estat la implantació d'una nova plataforma editorial. Fins aquell moment, els mitjans digitals i en paper utilitzaven sistemes totalment diferents, ja fossin productes comercials o propis, produïts internament o externa. Els principals eren Hermes i Gnoma en el cas del paper i DGrid per a les webs.

El nou sistema es coneix com Méthode i és comercialitzat per l'empresa italiana EidosMedia. El principal atractiu és la unificació de processos i continguts a l'hora de ser creats, per després poder ser utilitzats per diferents canals, web i paper. L'edició dels mateixos es realitza mitjançant un client instal·lat a la màquina de cada redactor, tot i que s'ha desenvolupat una versió online amb una funcionalitat encara molt limitada. Completen l'ecosistema una sèrie de serveis destinats a la introducció de continguts (principalment teletips d'agències de notícies, però també altres serveis contractats, com resultats esportius, passatemps, etc), gestió documental, exportació a les rotatives o al contenedor web.

Aquest últim té el nom de Méthode Portal Service (MPS), i en resum és un servidor Tomcat, modificat lleugerament, que llegeix continguts en XML des de la seva base de dades pròpia i genera HTML utilitzant plantilles JSP amb taglibs especialitzades.

En els últims dos anys i mig, s'ha implantat la plataforma i és utilitzada per La Vanguardia i Mundo Deportivo, tant en el paper com la web. S'han creat també les dues webs de bell nou, intentant treure factor comú d'ambdues tant com el disseny o les decisions de les diferents empreses han permès. Ha arribat el torn ara de la nova web de la ràdio RAC 1.

Fins ara, la web no estava especialment al dia comparada amb els seus competidors. Consistia en un Wordpress dividit en diferents seccions, una per programa de ràdio. La majoria d'elles no s'actualitzaven i les que sí ho feien només acostumaven a pujar un link als MP3 dels àudios amb una mica de text.

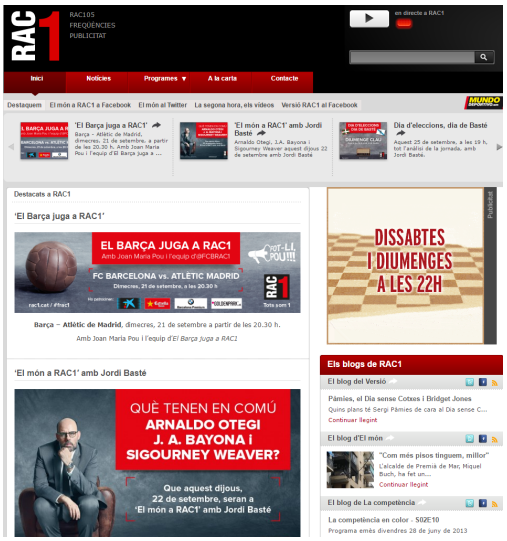

Figura 1: Web antiga: Portada

Figura 2: Web antiga: Pàgina de programa

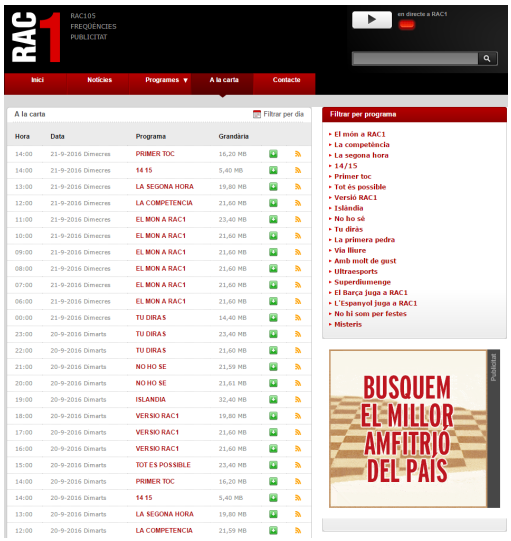

Figura 3: Web antiga: A la carta

La portada només era actualitzada un cop a la setmana, aproximadament, i a més els principals continguts no eren visibles en el primer scroll de la pàgina.

També tenia la secció d'àudios "a la carta". Des d'allà es podien escoltar o baixar el contingut sencer d'un programa o alguna de les seves seccions. Només era possible baixar continguts dels últims dies, però, i no tenia cap funcionalitat de cerca. Es tractava d'un desenvolupament propi realitzat en PHP, i allotjat a màquines externes a Grup Godó.

1.3 Objectius

La memòria d'aquest projecte no versa sobre el desenvolupament d'aquesta nova web, sinó només de la última part. El que fins ara es deia "a la carta"es coneix internament com Audioteca i com es pot suposar pel nom, la seva idea inicial era la d'una espècie d'hemeroteca a on es pugui trobar l'històric de totes les emissions. Com ja explicaré, l'aplicació tindrà moltes funcionalitats més.

Per entendre els objectius d'aquest projecte, primer hem d'entendre els de la nova web:

- Ha de ser una web de referència per a estar informat, no només un suport a la radio
- Tenir més contingut en text, notícies escrites, opcionalment amb algun àudio o fins i tot vídeo
- Presentació semblant a webs com la de La Vanguardia, amb una portada i seccions maquetable

- Ha de permetre treure més suc als àudios que la gent escolta. És a dir, poder traficar més publicitat
- Ha de permetre la generació de continguts mitjançant Méthode
- Disseny responsive, adaptat a desktop i mòbil
- Que el sistema estigui redundat, en alta disponibilitat entre dos CPD (Centre de Processament de Dades) diferents
- Seguir les mateixes directrius SEO de les webs de La Vanguardia i Mundo Deportivo, i utilitzar les mateixes llibreries per a la càrrega de publicitat (impressions), analítica, etc.

Els principals objectius de l'Audioteca són tot el que tingui que veure amb els fitxers d'àudio i la graella de programació:

- Integrar-se a dins la web de RAC 1, és a dir, per l'usuari serà una part de la web amb el mateix disseny
- Llistar els fitxers disponibles i proporcionar una eina de cerca
- Servir-los amb falques de publicitat inserides
- Proporcionar un player en JavaScript per escoltar-los
- Mostrar la graella de programes i mostrar quin és el programa que està sonant a cada moment

Per tal de fer tot això, la nostra empresa (Grup Godó) hem hagut de mantenir una comunicació constant amb altres empreses del grup: RAC 1 i Publipress.

2 Especificació

2.1 Anàlisi de requisits

Com he dit, Méthode proporciona el seu propi servidor web, el Méthode Portal Server. Per a cada site, es defineixen una sèrie de tipus de documents (notícia, secció, opinió, enquesta) i plantilles per renderitzar el contingut, JSP. Desgraciadament, no està pensat per a molt més que això i qualsevol intent d'estendre la seva funcionalitat és un *Via Crucis*.

Per tant, des de la creació de Mundo Deportivo (la primera web que vam fer amb Méthode) vam veure que necessitàvem una altra aplicació per generar l'HTML de les parts en que es requeria un contingut més dinàmic, obtingut des d'APIs de proveïdors, etc. En el cas de La Vanguardia i Mundo Deportivo, que tenen molta funcionalitat en comú, aquesta aplicació és la mateixa, i es diu Aside. Genera tot tipus de *peces*, en ocasions pàgines *pràcticament* senceres i en altres només uns pocs caràcters. Si dic *pràcticament* em refereixo a que mai volem generar tot el contingut d'una pàgina. El tag <head >i, en moltes ocasions el menú, footer, etc, no volem tenir-lo replicat en dos llocs (MPS i Aside). Per a mostrar una pàgina composta de parts generades pels dos, utilitzem Edge Side Includes (ESI), una tecnologia proporcionada per Akamai.

En el cas de la web de Rac1, les necessitats d'aquest contingut dinàmic no es semblen a les dels altres sites. Per tant, vam decidir fer l'Audioteca com una nova aplicació enlloc d'ampliar la funcionalitat de l'Aside. Com ja hem dit, la funcionalitat principal és la de gestionar els àudios. Però des del primer anàlisi vam veure que en realitat havia de resoldre molts més problemes:

- La principal funcionalitat és la de permetre pujar àudios al sistema en format MP3. Aquests àudios són de tres tipus:
  - **Talls horaris**. Es tracta de tot el contingut emès, d'hora en punt en hora en punt, dins els límits del programa. Per exemple, un programa que duri de 12 a 14, tindrà un tall de 12 a 13 i un altre de 13 a 14. Un programa que duri de 12:15 a 13:30 tindrà un tall de 12:15 a 13 i un de 13 a 13:30. Es tracta d'un procés automàtic realitzat per un agent que grava tot el contingut i el pujar mitjançant una API de l'Audioteca.
  - **Talls de seccions senceres**. Certs usuaris de l'Audioteca, anomenats tècnics, editen els àudios tallant on comença i on acaba una secció d'un programa i el pugen a través del backoffice.
  - **Píndoles**. S'anomenen així els àudios curts, habitualment d'uns segons a un parell de minuts, en que es destaquen parts d'una entrevista, un butlletí informatiu, etc. En un principi, aquest contingut no estarà disponible des de la secció "a la carta". Aquests talls els fan els redactors. Com que es vol que ho puguin fer tot des del client de Méthode, hem creat una API per integrar el client amb l'Audioteca.

Aquests àudios es modificaran per incloure en tags ID3 tota la informació relacionada: programa i secció del que formen part, data, títol, etc. El motiu d'això és que volem que siguin els mestres d'aquesta informació. En l'estrany cas que es corrompés l'índex o es perdés informació, podríem reindexar-ho tot de nou a partir d'ells. Una vegada modificat l'àudio es pujarà a un servidor extern, S3 d'Amazon, que proporciona alta disponibilitat i és visible des de fora. Finalment, tota aquesta informació ha de ser editable des del backoffice, per ampliar-la, corregir-la o eliminar l'ítem.

- Per tal de poder guardar a dins els MP3 de quin programa i de quina secció es tracta, l'Audioteca ha de mantenir un llistat de tots els programes i les seves seccions. De fet, també dels programes antics (no actius) que es puguin trobar des d'"a la carta". En realitat, l'Audioteca serà l'únic sistema que sabrà quines seccions té un programa, ja que a la resta de la web no es veuen. Per tant, des del backoffice s'ha de poder gestionar tot això.

- Hi ha certa informació dels programes, però, que sí es troba a Méthode. Per exemple la descripció. Per tal de no tenir la informació desincronitzada, necessitem que, quan es modifica a Méthode, s'actualitzti també a l'Audioteca.

- També ha de tenir la informació relativa a la graella de programació. És a dir, els horaris en que s'emeten tots els programes. No fa falta que en guardi tot l'històric, vam considerar que amb la graella de la setmana era suficient. Això s'edita també des del backoffice, proporcionant un sistema de plantilles per tal de no fer la feina massa farragosa. Aquesta informació serà utilitzada en diferents llocs:
  - Mostrar la graella a la web i a l'app
  - Saber quin programa està sonant a cada moment, per mostrar-ho també a la web i l'app
  - Proporcionar aquesta informació a l'agent que està gravant els talls horaris automàticament

- Player *estàtic*. Un dels principals requeriments de funcionalitat de la web era que en certes pàgines hi hagués un reproductor que en diem "estàtic". Es tracta d'un reproductor que, mentre l'usuari va navegant per les diferents pàgines, es troba reproduint un àudio o el directe sense talls. Això no ho permeten els navegadors, però és possible simular-ho. El truc és que, quan l'usuari clica a un link, es capturi l'event i es recarregui el contingut per Ajax. Mitjançant les operacions de JavaScript history.pushState() i history.onPopState() es pot simular el canvi de URL en la barra del navegador i els botons d'anar avant i enrere. Aquest contingut no pot contenir la pàgina sencera, sinó només el contingut que ha de canviar, sense capçalera, footer, etc. Com que això no és senzill, i per evitar problemes, només ho farem a les pàgines que genera l'Audioteca. Si en el futur veiem que no hi ha problemes, ho ampliarem a tota la web.

- Players de píndoles. Un redactor, en crear una notícia des de Méthode, pot voler afegir-hi una píndola. L'Audioteca és l'únic que coneix la informació d'un àudio (títol, ruta del fitxer MP3, etc). Amb la integració entre Méthode i l'Audioteca, el redactor pot buscar un àudio i Méthode en guardarà el seu id. Però Méthode no pot guardar la resta d'informació, ja que aquesta es pot canviar des del backoffice i no hi ha cap forma de notificar a Méthode en aquell moment. Per tant, mitjançant una inclusió ESI, quan es renderitzi la pàgina s'inclourà una crida a Méthode que, a partir de l'id de l'àudio, generarà un player. Aquest player té diferents estils depenent d'on es troba: la portada, un detall de notícia, etc.

- Els redactors permeten seleccionar una sèrie d'àudios i marcar-los com "essencials". Aquests essencials es llistaran a la portada, i poden ser desenes. Com que fer tantes inclusions ESI a la mateixa pàgina pot comportar problemes de rendiment, vam considerar millor opció que sigui l'Audioteca la que guarda quins àudios són essencials i sigui l'encarregada de generar aquest llistat.

- A més de la pàgina de la graella, també ha de generar pàgines amb diferents continguts:

  - Cerca d'àudios, filtrant per programa, secció o entre dues dates. També ha de permetre les cerques *full-text* pel títol, descripció o paraules clau. Això últim obliga a que, enlloc d'una base de dades relacional, s'hagi de guardar aquesta informació en un índex de text.
  - Pàgina de programa, on apareix informació bàsica del programa i el llistat dels últims àudios
  - Pàgina del directe, amb el que estigui sonant en aquest moment
  - S'ha de permetre compartir links a Facebook i Twitter. Per tant, fa falta una pàgina per a cada fitxer d'àudio, que serà a la que s'arribarà quan un usuari i faci clic des d'aquestes xarxes socials.

- Ha de proporcionar una API per tal d'alimentar l'app. Li proporcionarà informació que ja hem comentat: llistat de programes i seccions, els últims àudios de cada un, la programació, etc.

- També hem d'oferir un servei de podcast. En definitiva no és més que un RSS, però enlloc de links a pàgines HTML conté links als fitxers MP3. Hi haurà un podcast pels talls horaris de cada programa i per totes les seccions.

- Tots els àudios que estaven disponibles a la web antiga han d'estar disponibles també a la nova. Tot i que la gran majoria no eren accessibles (l'usuari no tenia forma d'arribar-hi des de la pàgina), el servidor contenia els àudios de l'últim any. Aquests seran els que migrarem en un primer moment. Per altre costat, durant el desenvolupament de la web també es migraran tots els continguts de Wordpress, i aquests poden tenir links a

fitxers MP3. Per tant, hem de tenir en compte que les dues migracions han *d'encaixar*. És a dir, que un post que té un link a un àudio determinat l'ha de mantenir a la nova web. S'ha decidit però, que no fa falta que es mantinguin les urls antigues dels fitxers MP3.

- Com que no podem obligar als usuaris de l'app antiga que se l'actualitzin, volem seguir mantenint compatibilitat durant un temps. És a dir, que els àudios que es pugin a l'Audioteca també s'han de copiar en el servidor antic.
- Inicialment, els àudios s'havien de servir mitjançant una plataforma d'inserció de publicitat. Com explicaré després, això no ha estat possible. Hem hagut d'implementar una solució provisional que fa el mateix.

2.2 Diagrama de seqüència

La majoria de processos són molt senzills a nivell de processos. Els més complexes, que impliquen vàries parts del sistema són la pujada d'àudios i servir-los a l'usuari. Aquest fluxe té un apartat propi a la secció d'implementació.

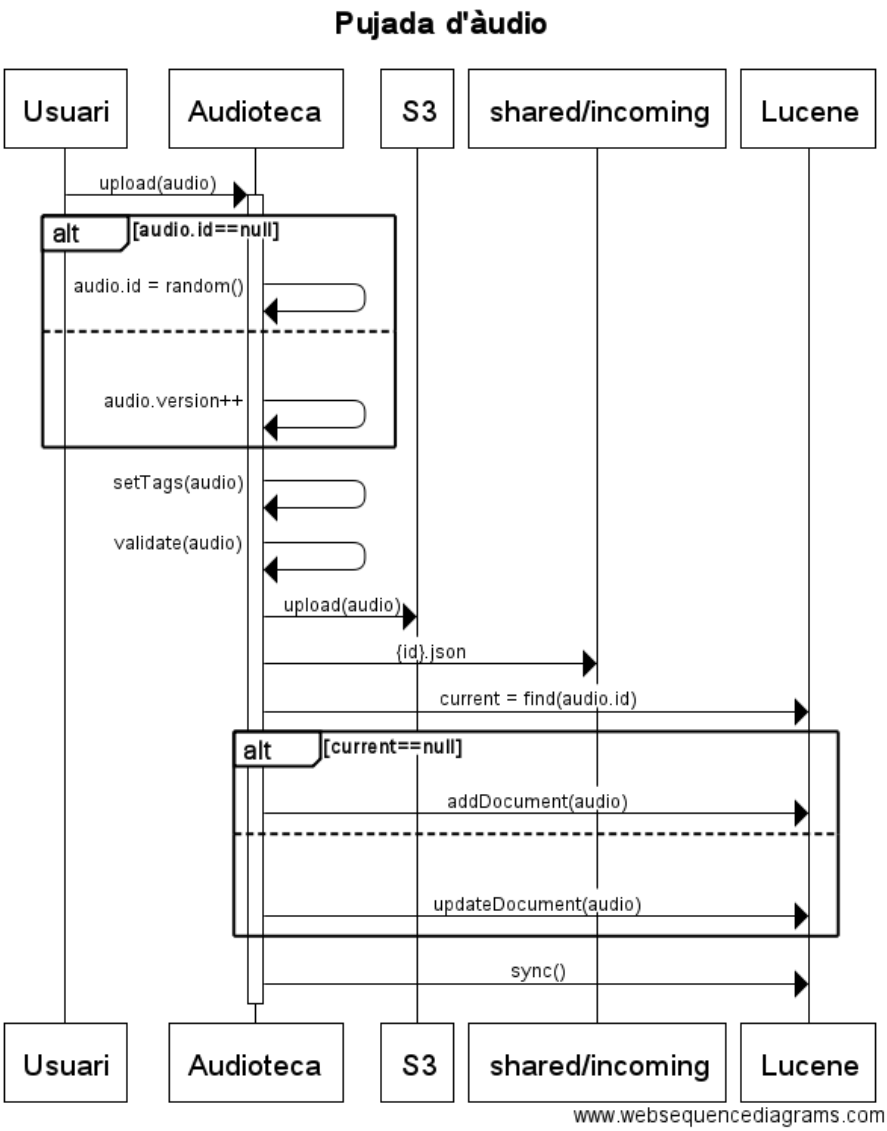

Figura 4: Diagrama de seqüència. Upload/update d'un àudio

3 Disseny

3.1 Arquitectura física

En el següent diagrama es poden veure tots els components i actors que entren en joc en el disseny de la solució:

- En rosat s'indiquen els components que s'han desenvolupat per aquest projecte
- Les fletxes indiquen cap a on viatge (generalment) la informació, no qui inicia aquesta comunicació
- No hi ha relació 1 a 1 entre els components i el número de servidors. De fet, la majoria d'aquests serveis necessiten vàries màquines, estan corrent en alta disponibilitat o fins i tot redundats entre dos CPD (Centre de Processament de Dades)

Actors:

- **Redactor:** Es tracta de personal de RAC1 dedicat a crear contingut per la web. Principalment notícies amb contingut escrit, però al tractar-se d'una ràdio, s'espera que moltes notícies tinguin talls d'àudio (especialment píndoles). Només utilitza el client de Méthode i no té accés a l'Audioteca. Per tant, per poder buscar àudios, modificar-ne les dades o pujar-ne de nous, ho farà des de Méthode i aquest es conectarà amb l'Audioteca per API.
- **Tècnic:** Es dedica a escoltar l'emissió per tal de crear talls de seccions senceres. A vegades també les edita per treure'n la publicitat. Té accés a l'Audioteca i hi puja els àudios a través del seu backoffice. També de RAC1.
- **Administrador:** Es tracta del rol de RAC1 amb més interacció amb l'Audioteca, i el més avançat dels usuaris. Des del backoffice, és l'encarregat de gestionar:
  - La informació dels programes i les seves seccions: títols, presentadors, imatges, identificadors de publicitat (que obté del backoffice d'AdsWizz), etcètera
  - La graella de la programació
  - Modificar els àudios ja existents o pujar-ne de nous
- **Traficker:** Es dedica a gestiona la publicitat que s'ha de sentir des en començar a escoltar l'streaming i els àudios a la carta, des de la web o l'app. Des del backoffice d'AdsWizz introdueix falques i campanyes publicitàries i defineix els criteris per decidir quina publicitat emetre. En aquests moments, els criteris són bastant limitats, ja que es redueixen a l'hora del dia i a la secció que es vulgui escoltar (en el cas dels àudios a

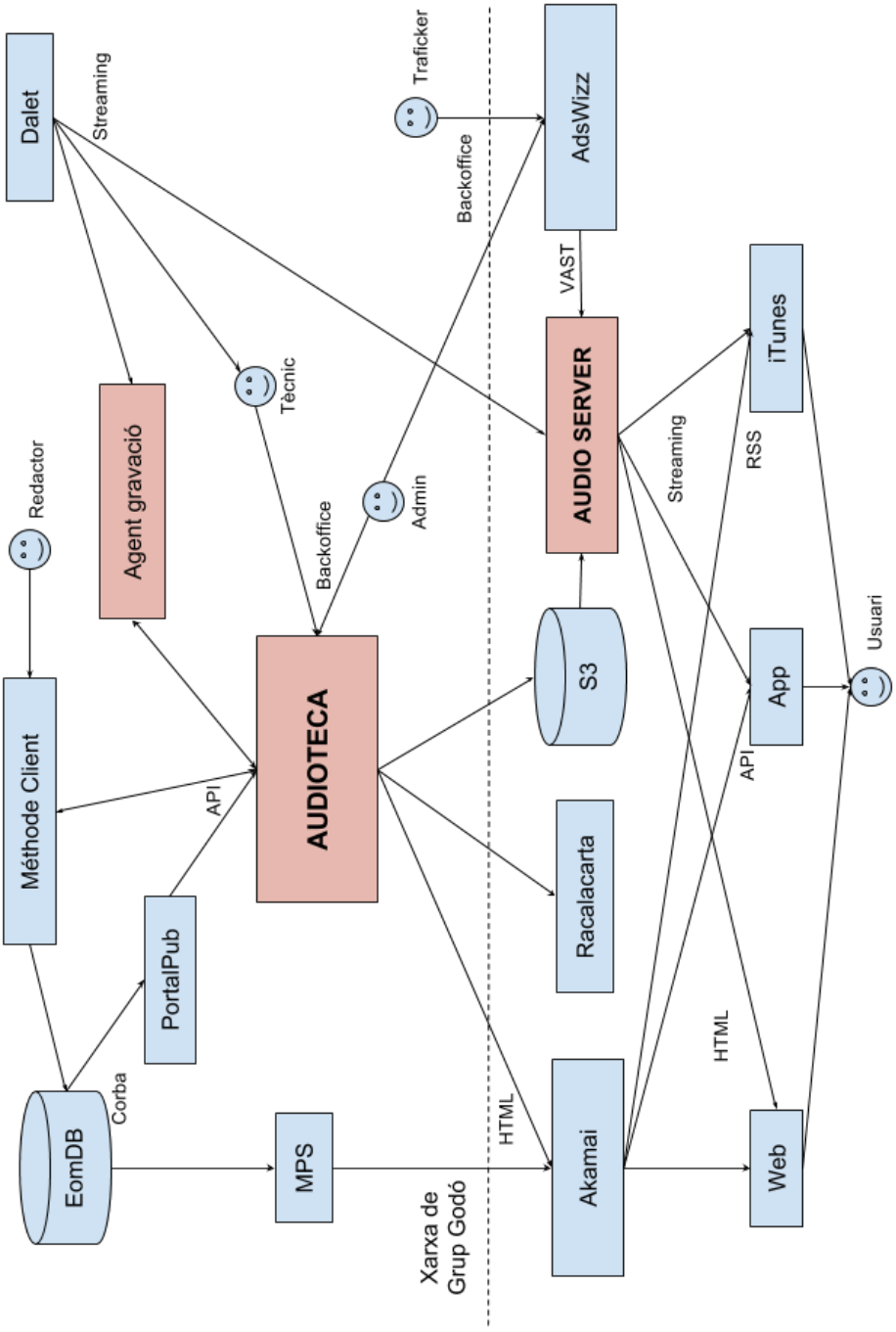

Figura 5: Diagrama de comunicació

la carta). Es tracta de personal de Publipress, l'empresa de publicitat de Grup Godó.

- **Usuari:** Fa referència a l'usuari final, que consumeix aquests continguts per tres vies:
  - Navegant per la web, podent escoltar els àudios i l'streaming des d'allà amb el player JavaScript
  - Utilitzant l'App de RAC1, des d'on també es poden escoltar els dos tipus, però no es poden buscar àudios relativament antics
  - Sindicant-se a un servei de podcast, com iTunes

Components:

- **EomDB:** La base de dades de Méthode. Conté les notícies que creen els usuaris o entren per agència, la distribució d'aquestes a dins la home o una secció, i fins i tot també guarda gran part de la configuració del sistema. Internament consta de tres parts:
  - Versant: una base de dades orientada a objectes, que conté les relacions entre les notícies o altres tipus de documents
  - Autonomy: un índex de cerca "full-text"
  - Un file system que conté tot tipus de documents en format XML i les imatges

  La comunicació amb altres components es fa mitjançant Corba
- **Méthode Client:** Es tracta del client pesat que utilitzen els redactors per escriure notícies i buscar entre el contingut ja generat. També pot dissenyar les pàgines dels diaris en paper i fins i tot enviar els PDF a la planta d'impressió, entre d'altres. Evidentment, en el cas de la ràdio, això no s'utilitza. No està pensat per poder incloure àudios, per tant hem hagut de desenvolupar noves funcionalitats perquè es pogués connectar amb l'Audioteca per pujar-hi àudios o buscar els que ja hi ha.
- **PortalPub:** És l'eina que es dedica a publicar les notícies als frontals web. En realitat, en el diagrama l'EomDB està una mica simplificada. Hi ha una instància per a les notícies que escriuen els redactors i una altra pel contingut que s'ha decidit publicar. PortalPub és qui transforma copia el contingut d'un lloc a l'altre. En aquest cas, hem fet que, cada vegada que des de Méthode Client es publiquen canvis sobre la informació dels programes o els àudios, actualitzi els canvis a l'Audioteca, mitjançant l'API.
- **MPS (Méthode Portal Server):** L'última de les tecnologies de Méthode que entra dins aquest projecte. És qui serveix el contingut de la EomDB en format HTML. Els seus fonaments són un Tomcat i plantilles JSP. Desgraciadament, es troba tant modificat que és complicat estendre'n la

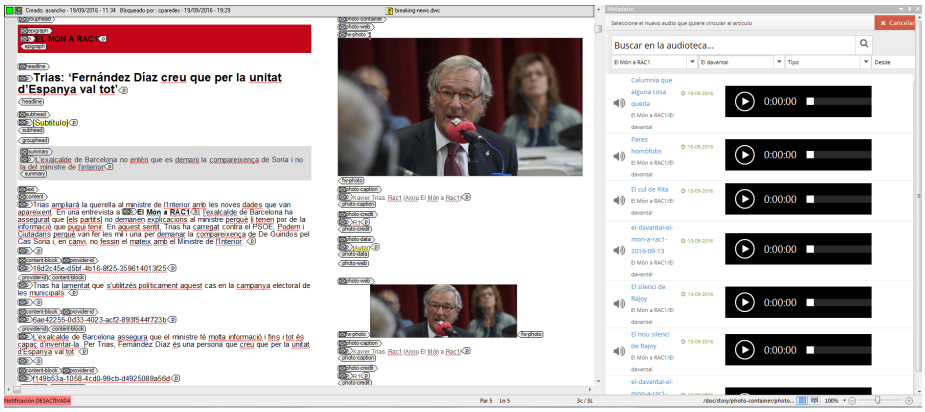

Figura 6: Méthode Client. Cerca d'àudios

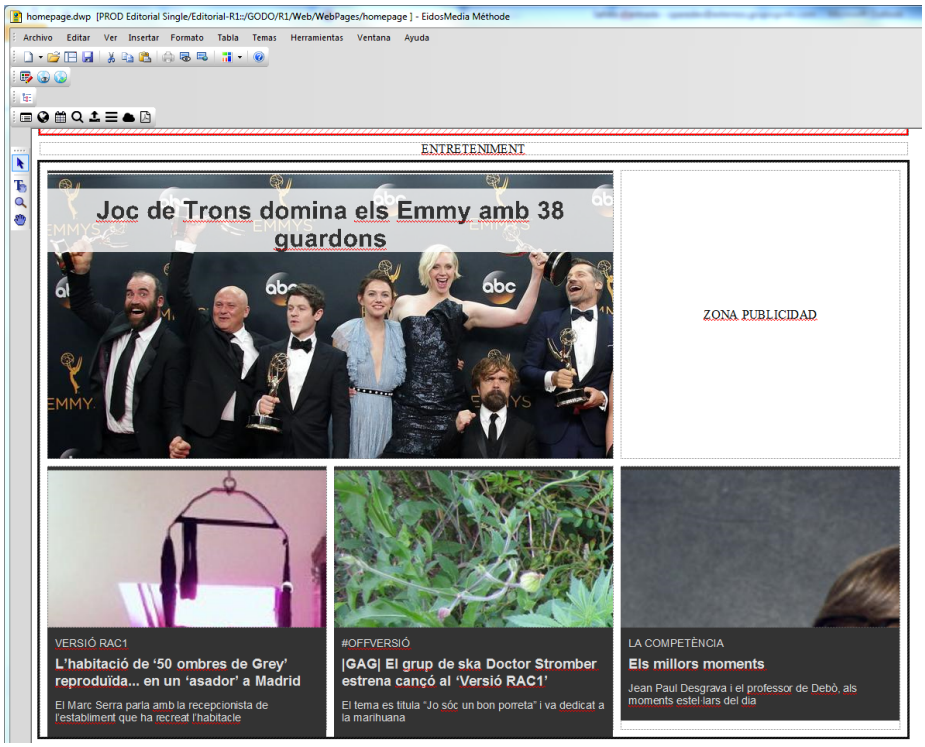

Figura 7: Méthode Client. Disseny de la portada

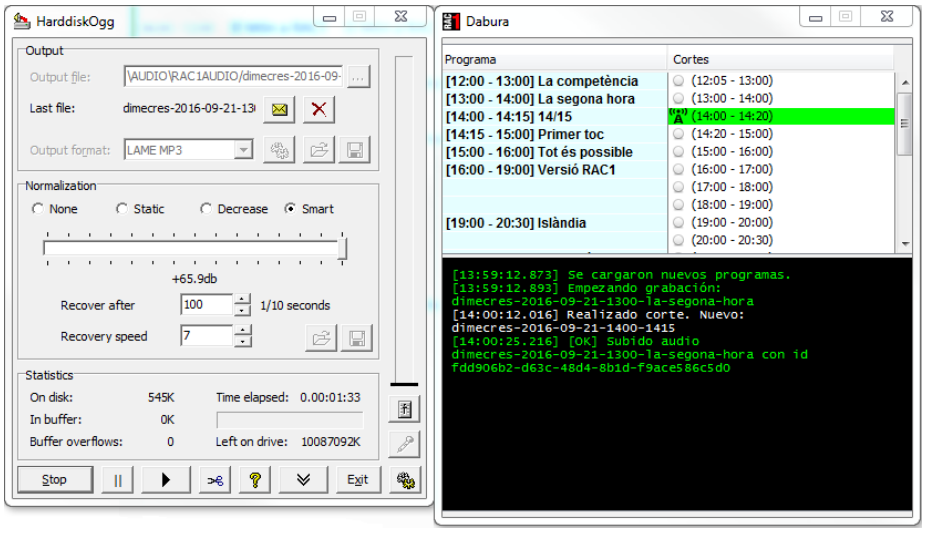

Figura 8: Agent de gravació

funcionalitat i resulta molt lent per a desenvolupar-hi. És per això que no es vol utilitzar per a generar les pàgines que finalment crea l'Audioteca.

- **Dalet:** Es tracta del sistema de gravació de l'àudio de la ràdio i també l'encarregat d'emetre en streaming.

- **Agent de gravació (Dabura):** Una petita aplicació desenvolupada en aquest projecte. Envia comandes a un servei en background (HarddiskOgg), que grava tot el que s'emet per Dalet. L'agent es conecta a l'API de l'Audioteca per obtenir la graella del dia i la informació dels programes, per saber en quin moment ha de realitzar els talls. Una vegada acabat un d'aquests fitxers, el publica a l'Audioteca.

- **Akamai:** La CDN (Content Delivery Network) que s'utilitza a la majoria de les webs del grup. Ens beneficia molt, ja que hem de respondre a molt menys volum de tràfic (un 20%) i minimitza l'impacte quan tenim la web caiguda. Una funcionalitat extra que no acostumen a tenir altres CDN és la que es coneix com ESI (Edge Side Includes). En aquest projecte l'utilitzem a davant tot el contingut de la web (HTML i recursos) i l'API per l'App. Pels fitxers d'àudio i l'streaming, en canvi, no resulta una bona opció. Principalment pel preu, ja que el cost de l'ample de banda és molt més alt que amb altres opcions. Durant la fase de test, on sí que es passava per Akamai de forma temporal, també hem vist que tenia greus problemes de rendiment per servir aquests fitxers, especialment quan rebia peticions demanant contingut parcial (capçalera HTTP Range).

- **Amazon S3:** Un servei d'storage massiu en el núvol, d'Amazon Web

Services. Té una disponibilitat molt alta i un preu per espai relativament baix. S'hi guarden els fitxers MP3 originals, és a dir, sense falques publicitàries.

- **AdsWizz:** Un servidor de publicitat de veu. Té un backoffice a on es poden crear campanyes i traficar les falques. També disposen d'una plataforma que s'encarrega de fer la fusió entre els MP3 originals i les falques, a més d'algunes funcionalitats molt interessants. Tal com explico més endavant (4.3.3), no s'ha pogut instal·lar la plataforma i ens comuniquem amb AdsWizz mitjançant crides VAST.

- **Audio Server:** Es tracta d'una solució provisional mentre no s'instal·li la plataforma d'AdsWizz al complet. És qui serveix tots els àudios al public (web, app o podcast). Obté els originals i els fusiona amb les falques publicitàries obtingudes d'Adswizz. Està replicat en diferents nodes allotjats a servidors externs a la xarxa de Grup Godó.

- **Audioteca:** És l'encarregada de gestionar la informació dels diferents fitxers d'àudio, els programes i les seves seccions i les graelles de programació. Tal com es veu en el diagrama, és el component amb més relacions, ja sigui amb sistemes automàtics o amb diferents rols.

- **Racalacarta:** Es tracta de l'antic servei de fitxers d'àudio, pel qual hem de mantenir compatibilitat.

- **App:** L'aplicació per a mòbils i tablets, en iOS i Android. Ja existia i durant aquest projecte, una empresa externa li ha fet un rentat de cara i l'ha modificada per obtenir les dades (agenda, llistat de programes, seccions i àudios, etc) de l'Audioteca via API.

3.2 Arquitectura lògica

Al departament intentem que totes les aplicacions tinguin un disseny semblant. La principal característica comuna entre totes ells és el clàssic disseny en tres capes. Aquestes són la de presentació, la de negoci i la de repositori (accés a dades), i cadascuna "parla"amb la següent mitjançant interfícies, sense saber res de la seva implementació. Els components de cada capa sí que poden conèixer la implementació dels seus veïns.

També tenim una altra capa, la del model. Aquesta és utilitzada per les altres capes. Aquesta és una decisió de disseny amb la que no tothom hi està d'acord. Hi ha qui advoca per que cada capa tingui el seu propi model, i conversors per passar d'un a l'altre. Es suposa que amb això cada capa pot tenir un model més adient a les seves necessitats. El principal problema d'això és que, depenent del llenguatge, això pot ser molt farragós. És el cas de Java, i això fa que mai ens haguem pres aquesta opció seriosament.

Amb l'auge del Domain Driven Design dels últims temps, a on els objectes de domini són molt més rics, alguns dels seus defensors, com Martin Fowler, han anomenat el disseny que nosaltres usem com "model de domini anèmic".

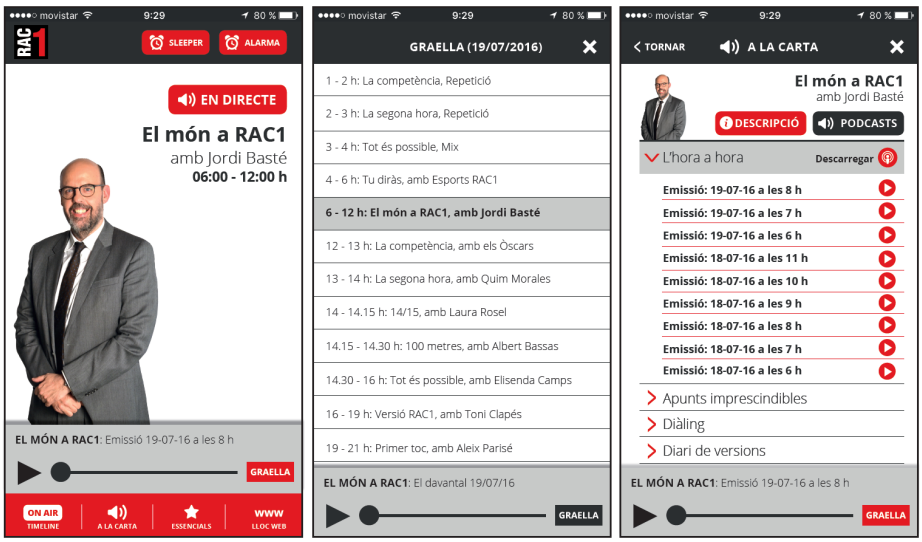

Figura 9: App (mobile iOS)

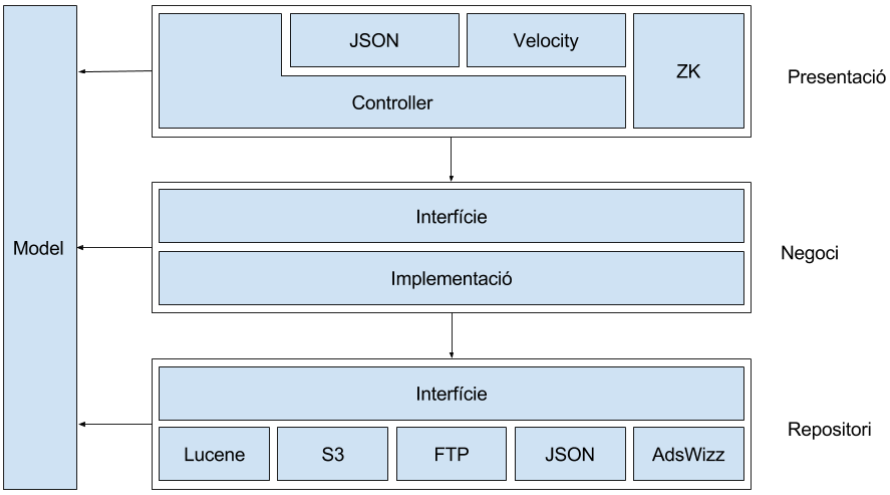

Figura 10: Disseny en capes

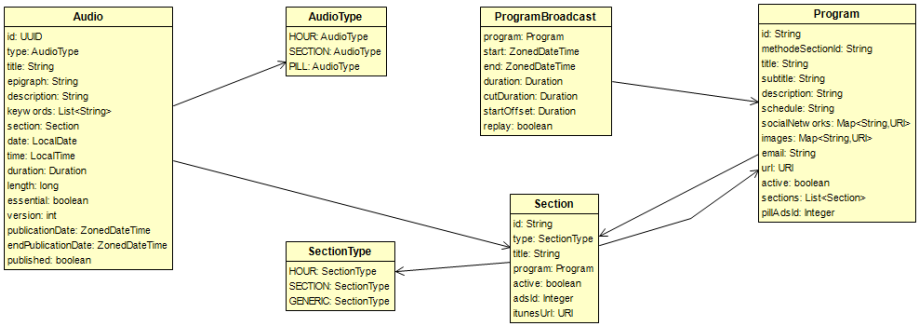

Figura 11: Entitats principals

En realitat, és un nom que a mi m'agrada. Crec que quan el model és realment minimalista, es queda en els ossos, i l'únic que conté són atributs i els seus "getters" "setters". D'aquesta forma queden objectes molt "purs", que no haurien d'estar contaminats per les necessitats de cadascuna de les capes. Per tant, pot succeir que en alguna ocasió sigui necessàries algunes entitats de model pròpies d'una capa concreta (generalment la de presentació, però alguna vegada també de repositori).

3.3 Model de dades

Aquest és el model de les principals entitats del model i la descripció dels atributs més importants:

- **Audio:** informació d'un tall d'àudio que es guardarà a l'índex (Lucene)
  - **id:** UUID generat aleatòriament quan s'introdueix un nou àudio a l'Audioteca. En el cas de la migració, com es veurà més endavant, aquest id no pot ser generat aleatòriament.
  - **type:**
    - HOUR: tall horari d'un programa
    - SECTION: secció sencera
    - PILL: "píndola", extracte d'un programa, generalment de pocs segons
  - **section:** tot àudio pertany a una secció. Si aquesta no és coneguda, s'assignarà la genèrica del programa (tots en tenen una)
  - **version:** Versió del binari. Només canvia quan des del backoffice es puja un nou mp3 per un àudio ja existent. Com que els servidors d'àudio tenen cachés dels fitxers i no tenim forma de comunicar-ho des de l'Audioteca, necessitem aquest camp per saber que s'ha de baixar una nova versió. Si ens integrem amb la plataforma d'Adswizz probablement no tindrà sentit.

  - **publicationDate, endPublicationDate:** Les notícies a Méthode permeten guardar-se amb una data de publicació futura. Això significa que no serà visible pel public fins aquell moment. De la mateixa forma, a vegades les notícies caduquen o s'eliminen especificant una data de fi de publicació. Aquest mateix comportament es reprodueix també a l'Audioteca
  - **published:** Quan un àudio es puja a l'Audioteca des de Méthode no està publicat al moment. Està en un estat visible només per als usuaris interns. Quan es fa una publicació explícita és quan sí es visible per a tothom.

- **Program:** representa un programa de ràdio, no la seva emissió
  - **id:** identificador usat per a crear URLs i rutes dels fitxers. En minúscules, guions i sense espais ni caràcters estranys: el-mon, la-competencia, etc
  - **methodeSectionId:** id del programa a Méthode, ja que no hi ha una relació 1 a 1. Per exemple: /programes/el-mon
  - **title:** nom del programa: El món a RAC1
  - **images:** mapa amb totes les imatges d'un programa que es necessitaran des de la vista
  - **active:** per tal de mantenir l'històric dels àudios no podem deixar que s'esborri cap programa. Però necessitem aquest booleà degut a que en certs llocs no s'han de mostrar els que ja no s'emeten més.
  - **sections:** llistat de seccions

- **Section:** secció d'un programa, ja sigui real o virtual (talls horaris o genèrica)
  - **id:** similar a l'id dels programes
  - **type:** una secció pot ser de tres tipus (que no es corresponen al 100% amb els tipus dels àudios):
    * HOUR: secció que conté només els àudios de tipus HOUR
    * SECTION: conté els àudios de tipus SECTION de la secció en qüestió, així com les "píndoles"(PILL) d'aquella secció
    * GENERIC: conté àudios de tipus SECTION i PILL de programes coneguts, però que no pertanyen a una secció determinada. Pot ser degut a tres casos:
      · píndoles que no pertanyen cap secció, com pot ser el butlletí informatiu
      · seccions especials que no s'ha considerat oportú crear. Per exemple, un especial de Nadal
      · àudios dels quals no se n'ha pogut obtenir la secció durant la migració, parsejant-ne el nom del fitxer

  - **adsId:** Id de la secció en el proveïdor de publicitat (AdsWizz)

- **ProgramBroadcast:** representa l'emissió real d'un programa. Amb una data d'inici i fi d'un dia concret
  - **program:** El programa al que fa referència
  - **start, end:** instants d'inici i fi. S'utilitza la classe ZonedDateTime de Java 8, que a més d'una data i una hora conté el desplaçament en relació a l'hora GMT (per exemple *2016-09-20T07:37:31.281 +02:00[Europe/Paris]*). A les vistes, hi ha vegades que necessitem mostrar l'hora del programa en el nostre fus horari (per exemple, a la pàgina de la graella, on no sabem des d'on ens estan visitant). En canvi a l'app sí que podem saber on està la persona i mostrar-li les hores locals. La classe ZonedDateTime ens permet obtenir tota aquesta informació en un sol tipus.

4 Implementació

4.1 Entorn de desenvolupament

4.1.1 Repositori de codi

En aquests moments estem utilitzant Subversion. S'està migrant tot a Git i el plan és que ens hi passem tots els desenvolupadors en unes setmanes. Si no ho fem de cop, tindrem bastants problemes, ja que tenim vàries llibreries pròpies compartides en tots els projectes.

4.1.2 Maven

És l'eina de gestió de dependències i de construcció del projecte. També tenim un servidor Artifactory, un contenidor privat amb les nostres llibreries i altres dependències requerides.

4.1.3 Jenkins

És el servidor d'integració continua (CI). Es conecta al repositori de codi, descarrega les dependències d'Artifactory, compil·la el codi, passa els tests i, en cas d'èxit, munta el projecte. El .war resultant, així com el codi font i la documentació és pujat a Artifactory.

4.1.4 IDE

L'entorn integrat de desenvolupament que utilitzem és una versió personalitzada d'Eclipse: Godoclipse. En aquests moments estem actualitzats a la última versió d'Eclipse, Neon. No té res d'especial, però està pensat perquè qualsevol que se'l descarregui ho tingui tot preparat per treballar en pocs segons:

- Té instal·lats els plugins de Subversion, Maven, Project Lombok, etc
- Una configuració comú, per estar segurs que, per exemple, un formateig del codi, no implica molts canvis degut a que dues persones tenen regles diferents
- Els repositoris de codi ja configurats
- Accés a Méthode
- Té configurats diferents Tomcats, amb els ports que utilitzen els diferents projectes en el mode de desenvolupament

Pot resultar estrany que tinguem diferents Tomcats instal·lats, quan sembla que amb un hauria de ser suficient. Per provar en local la web de RAC1 amb l'Audioteca en fan falta 3:

- La web s'executa a dins el seu propi MPS en local. El problema és que no admet cap altra aplicació corrent-hi a dins

- Tenim un altre Tomcat amb l'Audioteca
- Finalment, un tercer que només conté un projecte, el web-preview. Es tracta d'un projecte que assoleix diferents objectius:
  - És a on es creen les maquetes dels nous sites. Mitjançant plantilles en Velocity, els maquetadors generen un HTML semblant al final
  - Genera els recursos CSS i JS, a més de contenir els recursos estàtics. Si volem veure la maqueta necessitem que aquests recursos estiguin aquí. A més, els CSS es generen a partir de LESS i els JS passen per un procés de fusió i minimització. Dificilment podríem fer això en l'MPS.

  Per provar la majoria de parts de l'Audioteca no ens fa falta tenir els altres dos en marxa, però si volem veure l'Audioteca a dins el marc de la web de RAC1, els necessitem.

4.2 Principals tecnologies

En aquest apartat s'enumeren les principals tecnologies que s'utilitzen pel funcionament de l'Audioteca.

4.2.1 Java

Intentem que totes les aplicacions que desenvolupem siguin relativament semblants. I la primera decisió al respecte és la de sempre utilitzar Java. Això també permet que l'entorn de desenvolupament sigui sempre el mateix, com hem vist en el punt anterior. El desplegament de les aplicacions també és idèntic en quasi tots els casos: desplegar un .war a dins una instància de Tomcat.

La principal novetat al respecte en aquest projecte ha estat que es tracta del primer en utilitzar Java 8. Personalment, es tracta de la tecnologia que més ganes em feia utilitzar. Ja portàvem anys de retràs respecte l'estat de l'art, d'aquest i d'altres llenguatges, ja que Java 7 s'havia quedat molt antiquat. Si no ho hem fet abans ha estat degut a que el servidor MPS només funciona amb Java 7. Evidentment, el codi de l'Audioteca no té relació amb l'MPS, però això ens obliga a que les llibreries nostres que utilitza l'MPS també estiguin en Java 7. En definitiva

De l'ús que en fem en aquest projecte en destacaria els següents punts:

- **Lambdas**: Es tracta de funcions anònimes que es poden passar com argument a un mètode o com a retorn d'aquest. Molt relacionat amb les lambdas, i també com a *novetat*, hi ha les *MethodReference*, que permeten obtenir un mètode (que es pot passar també com argument) a partir del seu nom. Això permet escriure un codi més net, especialment en el cas de les operacions sobre llistes o altres col·leccions.

  Per exemple, aquí hi ha dues versions d'un mateix mètode que retorna una nova llista amb strings sense espais als costats i sense elements buits. Com es veu, en el segon cas és més curt i més fàcil de llegir.

```java
// Amb Java 7
public static List<String> clean7(List<String> strings) {
    List<String> list = new ArrayList<>();
    for (val s : strings) {
        String trimmed = s.trim();
        if (StringUtils.isNotBlank(trimmed)) {
            list.add(trimmed);
        }
    }
    return list;
}

// Amb Java 8
public static List<String> clean8(List<String> strings) {
    return strings.stream().map(x -> x.trim()) // Lambda
            .filter(StringUtils::isNotBlank) // Method reference
            .collect(Collectors.toList());
}
```

- **java.time**: Les classes que fins ara proporcionava l'API estàndard de Java pel tractament de dates (*java.util.Date* i *java.util.Calendar*) eren molt limitades i propenses a produir errors. Ja feia anys que molts desenvolupadors les evitaven i utilitzaven *Joda time*. El seu mateix creador, Stephen Colebourne, ha estat l'encarregat de crear una nova implementació millorada per a l'API estàndard. El package *java.time* té els següents avantatges:
  - Conté classes diferents per a representar dates, hores, dates amb hora, dia de la setmana, durades, etc. D'aquesta forma, permet un codi molt més *type safe*: a partir del tipus ja es pot determinar quina informació representa. Abans, enlloc de tot això només teníem enters (un *java.util.Date* només és un wrapper d'un *long*), i per tant s'havia d'especificar de quina informació es tractava afegint documentació.
  - Conté classes diferents per a representar hores humanes i punts determinats en el temps. No és el mateix dir l·les tres del migdia" (*LocalTime*), que l·les tres del migdia del 3 de març" (*LocalDateTime*) o l·les tres del migdia del 3 de març en la zona horària de Noruega" (*ZonedDateTime*). Només aquesta última representa un punt inequívoc en el temps.
  - Proporciona diverses formes de convertir al format ISO 8601 i a la inversa. Aquest format és el que hem utilitzat per a totes les APIs. D'aquesta forma la integració amb altres llenguatges (com és el cas de l'app) resulta molt més directa.

4.2.2 Spring

Spring és d'ús pràcticament obligat en tots els nostres desenvolupaments. Concretament, en aquesta aplicació n'hem utilitzat les següents parts:

- Spring Framework: el core, al voltant del qual orbiten tots els altres paquets d'Spring. La peça central és un contenedor de beans, que permet la injecció de dependències (DI, una forma d'inversió de control) i la orientació a aspectes (AOP).
- Spring MVC: com el seu nom indica, implementa el patró Model-Vista-Controlador. Permet declarar operacions dels controladors d'una forma

molt neta, i habitualment no és necessari accedir als objectes *HttpServletRequest* i *HttpServletResponse*.

- Spring Security: proporciona un registre central per l'autenticació i després permet autoritzar l'accés de diferents formes: amb patrons d'URL, amb anotacions a sobre mètodes, amb funcions a la vista, etc. En aquesta aplicació només hem optat pel primer mètode.

4.2.3 Lucene

Es tracta d'un índex per a realitzar cerques *full text* de documents. Per a fer-ho, quan s'indexa un document, un analitzador sintàctic realitza una sèrie de transformacions (dividir-lo en paraules [tokenització], eliminar les poc importants [stop words] i extreure'n l'arrel [stemming]). Cada paraula és després indexada guardant a quins documents apareix. Quan es fa una cerca d'un text, aquest pateix la mateixa transformació i es busca en quins documents apareixen cadascuna de les seves paraules. Aquesta cerca resulta molt més ràpida del que podria ser amb una base de dades relacional. El costat negatiu, és que no hi pot haver relacions entre documents.

Es tracta d'una peça central en el nostre desenvolupament. Guarda tota la informació dels àudios (excepte el propi binari), fins i tot la seva durada o la seva mida en bytes. És l'eina que utilitzem per a fer les cerques dels mateixos.

4.2.4 Edge Side Includes (ESI)

Com ja he comentat, en molts casos hem de generar una sola pàgina web que està formada per parts servides des de l'MPS i l'Audioteca. Davant els nostres frontals web hi tenim Akamai com a CDN. Una de les funcionalitats opcionals més interessants que té és la dels Edge Side Includes. Com el seu nom indica, consisteix en inclusions realitzades per la CDN (Edge). Per fer això, una vegada activat, parseja tots els documents HTML buscant expressions de l'estil *<esi:include src=ürl-a-afegir"/>*. Si no té el contingut en caché, fa una petició per obtenir-lo i el fusiona l'HTML original:

- Les parts inserides han de retornar unes capçaleres HTTP de caché. Això permet que pàgines que es modifiquen amb molt poca freqüència continguin fragments més dinàmics i no s'hagin de fer peticions contínuament per obtenir tot el contingut. Per exemple, en el nostre cas, quasi totes les pàgines mostren quin és el programa que s'està emetent ara. Si el temps de refresc de cada pàgina hagués de ser de, per exemple, un minut, tindríem la caché pràcticament buida.

- S'ha d'anar en compte amb els errors quan es fa la crida del segment inclòs. Es poden utilitzar *try/catch*, però en el cas d'un 404, per exemple, es retornara la pàgina contenidora sense la inclusió. L'usuari, llavors, no rebrà un 404.

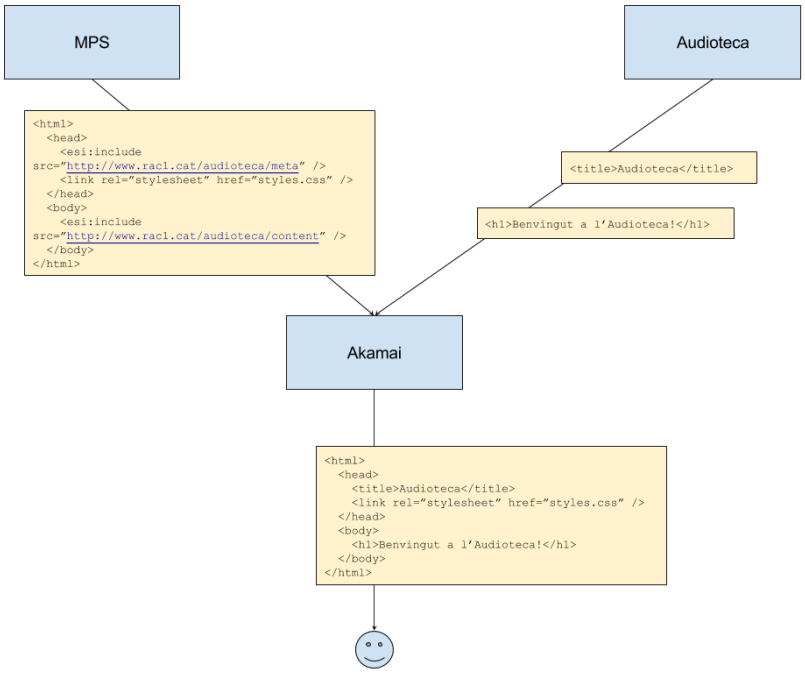

Figura 12: Edge Side Includes

Es tracta d'un estàndard, però només Akamai l'implementa al 100%. La principal funcionalitat (*include*) també és implementada per alguna altra CDN (Fastly) o Varnish, un servidor de caché pensat per a realitzar diferents transformacions en servir un contingut web a un usuari.

4.2.5 Project Lombok

Es tracta d'una llibreria relativament desconeguda, però d'aquelles que una vegada l'has utilitzada ja no te'n pots desprendre. D'una forma aparentment màgica, transforma el codi font en el moment de la compil·lació per tal d'el·liminar certs tipus codi repetitiu (boilerplate). El truc està en utilitzar APT (Annotation Processing Tools), una funcionalitat poc coneguda de Java per tal de crear altre codi o realitzar validacions durant la fase de compil·lació. Amb una sèrie d'anotacions permet, per exemple, generar *getters* i *setters*, mètodes *toString()*, *equals()* i *hashcode()* entre moltes altres transformacions. És molt útil en tot el codi, però les classes de model, que volem que siguin molt senzilles.

Per exemple, el codi font:

```
1  @lombok.Data // Indica que volem transformar la classe
2  public class X {
3
4      private String y;
5  }
```

Genera un bytecode similar a :

```
1  public class X {
2
3      private String y;
4
5      public String getY() {
6          return y;
7      }
8
9      public void setY(String y) {
10         this.y = y;
11     }
12
13     public String toString() {
14         return "X(y=" + y + ")";
15     }
16
17     public boolean equals(Object o) {
18         return (o instanceof X && y.equals(((X) o).y));
19     }
20
21     public int hashCode() {
22         return y.hashCode();
23     }
24 }
```

4.2.6 Velocity

Es tracta d'un sistema de templates. L'utilitzem per generar la vista de les pàgines de l'Audioteca (a la carta, graella, etc).

4.2.7 ZK

És una altra llibreria per a la capa de presentació. En aquest cas, enlloc del patró petició HTTP -¿ resultat, utilitza un model basats en components, com Java Server Faces. Proporciona molts components habituals ja creats (popup, combobox, etc). No està pensat per servir contingut de forma massiva: es costós, acostuma a mantenir l'estat en el servidor, etc. Però ens va molt bé per la part del backoffice. Un desenvolupador acostumat a l'eina pot crear un backoffice com el de l'Audioteca en qüestió de dos o tres dies.

4.2.8 Mp3agic

Probablement el que més m'ha sorprès durant el desenvolupament ha estat el poc suport per mp3 que existeix en el món Java. Per tal de fer els audioServers ens hauria anat molt bé una eina capaç de fusionar dos fitxers. No en vaig trobar cap.

Una altra necessitat que teníem era la de llegir i escriure tags ID3 a dins els MP3. Hi ha diferents llibreries que ho permeten, però totes elles tenen algunes limitacions. A més, semblen realitzades per una única persona de forma amateur i en general són bastant antigues. Després de provar-ne vàries em vaig quedar amb Mp3agic. Tot i ser la preferida, té un parell de coses que no m'agraden:

- Totes les operacions es fan sobre fitxers, no hi ha cap opció d'utilitzar un array de bytes o Input/OutputStream
- No té l'operació de guardar a sobre del mateix fitxer que ha obert per llegir, sempre que llegeix d'un fitxer és per escriure a un altre.

Aquestes dues coses fan que a l'Audioteca, on seria més normal operar amb arrays de bytes, hagi d'estar passant-me còpies de fitxers. A més, vaig trobar un bug (menor) del qual encara no he rebut resposta.

4.2.9 Validator

Java proporciona un estàndar de validació basat en anotacions. Aquestes representen restriccions i es col·loquen a sobre dels atributs de la classe. Quan en volem validar una instància li passem al validador, que ens dona un llistat de violacions.

```
1  @Data
2  public class Audio implements Identifiable<UUID> {
3
4      @NotNull
5      private UUID id;
6
7      @NotNull
8      private AudioType type;
9
10     @NotNull
11     @NotEmpty
12     private String title;
13
14     @NotNull
15     @Valid // Valida l'entitat relacionada
16     private Section section;
17
18     /**
19      * Mida del fitxer original en bytes
20      */
21     @Min(100)
22     private long length;
23
24 }
```

4.2.10 jUnit

És el framework de testeig per excel·lència de Java. El seu nom sembla indicar que està destinat a tests unitaris, i així era en un principi, però actualment pot fer molt més. Ens serveix per a realitzar dos tipus de test:

- Unitaris: en que només es prova una única classe
- D'integració: en que més d'un component és testat de cop. Tot iq ue en fem dels dos tipus, aquests són els que més predominen en el nostre codi. Hi ha qui creu que han d'abundar els unitaris, però basant-nos en la nostra experiència, veiem que invertint el mateix esforç, provem més factors i trobem més errors amb aquests últims. A més, jUnit s'integra bastant bé amb Spring. És possible arrencar el context i injectar-te els beans que vols provar directament a la classe de test.

La configuració d'Spring la fem en capes. És a dir, tenim una classe que declara tots els beans de repositori, una altra de servei i una altra per la presentació. Quan executem els tests amb jUnit, es pot indicar quin és el fitxer de configuració que agafem. D'aquesta forma, quan fem un test de la part de repositori estem comprovant també que no hi hagi accesos il·legals entre beans: per exemple, si un repositori vulgués accedir a un bean de servei donaria un error per no estar declarat.

Com a exemple de test d'integració, tenim aquest, bastant complet, on realitzem el següent:

- Arrenquem el contexte d'Spring. Com hem dit, configurem Spring per capes, i en aquest cas, demanem la capa superior, @WebConfig, que implica tots els beans.
- Fem operacions sobre tot l'MVC d'Spring. L'únic que no estem provant aquí és el propi Tomcat.
- Provem l'API d'integració amb Méthode, concretament l'operació de pujada d'un àudio, proporcionant totes les dades que es necessiten, algunes amb caràcters estranys
- Es valida aquesta informació i s'introdueix en ID3 al fitxer
- Es puja l'àudio a S3 (una instància de proves)
- S'actualitza l'índex
- Es valida que la resposta del controlador és correcta (els camps coincideixen)
- S'executa una altra operació del controlador per cercar aquest fitxer i es valida el resultat
- Es demana la informació d'aquest fitxer. Un dels atributs del resultat ha de ser la URL d'S3 del fitxer físic.
- Intentem accedir al fitxer i veiem que ens retorna un 200 (èxit)

```
1  @WebAppConfiguration
2  @ContextConfiguration(
3      classes = { WebConfig.class, TestConfig.class })
4  public class MethodeExtensionsApiControllerTest extends AbstractSpringTest {
5
6      @Autowired
7      private XsonFactory xson;
8
9      @Test
10     public void uploadSearchGet() throws Exception {
11
12         /**
13         * Pujada de l'audio
14         */
15         val testFile = new File("src/test/resources/samples/summertime.mp3");
16         val today = LocalDate.now();
17         val now = LocalTime.now();
18         val res = getMvc().perform( //
19                 fileUpload("/api/methode/extensions/v1/upload") //
20                         .file("file", FileUtils.read(testFile)) //
21                         .param("type", AudioType.SECTION.toString()) //
22                         .param("programId", "el-mon") //
23                         .param("sectionId", "entrevista") //
24                         .param("title", "The title of this audio file") //
25                         .param("keywords", "x, Latex no mostra chars estranys")
26                         .param("description", (String) null) //
27                         .param("date", today.toString()) //
28                         .param("time", now.toString()) //
29         ).andExpect(status().isOk()).andReturn();
30         val audio = xson.create(res.getResponse().getContentAsString()) //
31                 .as(Audio.class);
32
33         assertNotNull(audio.getId());
34         assertNotNull(audio.getDuration());
35         assertEquals("el-mon", audio.getSection().getProgram().getId());
```

```
36          assertEquals("entrevista", audio.getSection().getId());
37          assertEquals(today, audio.getDate());
38          assertEquals(now, audio.getTime());
39          assertEquals(Arrays.asList("x", "Latex no mostra chars estranys"), audio.getKeywords());
40
41          /**
42           * Executar la cerca
43           */
44          val res2 = getMvc().perform( //
45                  get("/api/methode/extensions/v1/search") //
46                          .param("type", AudioType.SECTION.toString()) //
47                          .param("programId", "el-mon") //
48                          .param("sectionId", "entrevista") //
49                          .param("text", "audio") //
50                          .param("from", today.minusDays(1).toString()) //
51                          .param("to", today.plusDays(1).toString()) //
52                          .param("onlyPublished", "false") //
53                          .param("pageSize", "10") //
54          ).andExpect(status().isOk()).andReturn();
55          val audioListType = new TypeToken<List<Audio>>() {}.getType();
56          List<Audio> list = xson.create(res2.getResponse().getContentAsString()) //
57                  .get("content") //
58                  .as(audioListType);
59
60          assertTrue(!list.isEmpty());
61
62          /**
63           * Executar el metode get i comprovar que es retorna un path valid a l'mp3
64           */
65
66          val res3 = getMvc().perform( //
67                  get("/api/methode/extensions/v1/get/" + audio.getId()) //
68          ).andExpect(status().isOk()).andReturn();
69          val audioView = xson.create(res3.getResponse().getContentAsString()) //
70                  .as(AudioView.class);
71
72          val res4 = new HttpApiClient().createRequest(audioView.getPath().toString()).get();
73          assertEquals(200, res4.getStatus());
74          assertTrue(res4.getContent().length > 1000);
75      }
76
77  }
```

4.3 Parts destacades

4.3.1 Sincronització Lucene

Un dels requisits de la majoria d'aplicacions que es fan a Grup Godó és l'alta disponibilitat. Per algunes, a més, també es vol que estiguin replicats en dos CPD (Centre de Processament de Dades) o Data Centers. El principal problema, doncs, no és que l'aplicació hagi d'estar dissenyada per l'alta disponibilitat, sinó que molts recursos, com ara una base dades, no sigui possible tenir-los sincronitzats, o les solucions no siguin ni barates ni senzilles a nivell d'infraestructures.

Això també és un inconvenient per Lucene. La primera opció que se'ns va passar pel cap va ser que l'índex fos compartit en una carpeta NFS (Network File System) entre els dos CPD. Per tal com està fet, Lucene només pot escriure a un índex a la vegada. Per tant, igualment s'hauria hagut de buscar alguna forma de sincronitzar els nodes, de tal forma que sempre només un d'ells fos el que rep les peticions d'escriptura. A més, el rendiment es veu molt sèriament afectat.

Una altra opció a tenir en compte va ser la d'utilitzar Solr. Es tracta d'un servidor standalone que utilitza Lucene. Els seus majors avantatges són l'escalabilitat i la tolerància a fallades. El problema és que el suport per a la sincronització entre diferents data centers (CDCR: Cross Data Center Replication) és molt recent, del març d'aquest any. No he pogut trobar moltes opinions

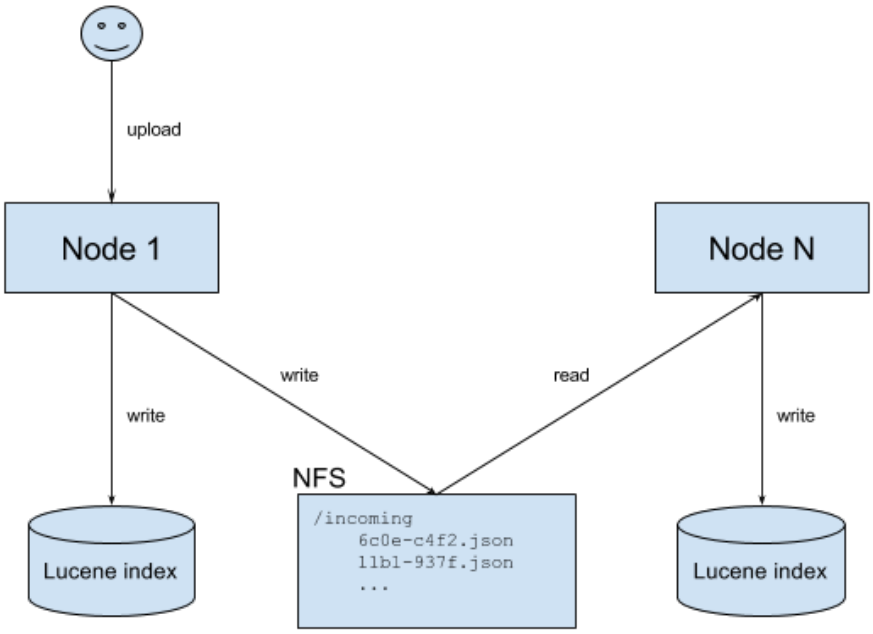

Figura 13: Sincronització de Lucene

al respecte. Per altre costat, aquest tipus de solucions suposa un problema per l'equip d'infraestructures. Per la majoria de projectes, procurem que per desplegar una aplicació, més enllà de donar permisos i obrir ports, sigui suficient en desplegar el fitxer .war.

Després d'estudiar les diferents alternatives, per tant, i ponderar tots els pros i contres, vam optar per una solució diferent. Vam decidir que cada node tindria la seva carpeta local amb un índex de lectura i un d'escriptura. Per tal de sincronitzar la informació dels nodes, utilitzem una carpeta NFS a on guardem fitxers amb la informació de cada àudio, en format JSON. Tots els nodes monitoritzen aquesta carpeta per actualitzar els últims canvis en el seu índex local.

És a dir, quan es genera un àudio, per exemple amb id 14, es genera un fitxer 14.json amb el següent contingut:

```
{
  "id": "06c0e421-9387-3e54-840e-7d1df83ec4f2",
  "type": "SECTION",
  "title": "musica-el-mon-a-rac1-2016-09-07",
  "keywords": [],
  "section": {
    "id": "musica",
    "program": {
      "id": "el-mon"
    }
  },
  "date": "2016-09-07",
  "time": "11:00:00",
  "duration": "PT11M35S",
  "length": 4182664,
```

```
16    "essential": false,
17    "version": 1,
18    "publicationDate": "2016-09-07T00:00:00+02:00",
19    "published": true
20  }
```

4.3.2 Dades persistides

El mateix problema de la distribució de Lucene entre dos CPD ocorre amb les dades persistides. És a dir, necessitem algun lloc a on guardar la informació relativa als programes i seccions i les graelles de programació. Una base de dades seria el recurs habitual. De fet, Grup Godó disposa d'un sistema Exadata d'Oracle. És una solució combinada que proporciona gran capacitat y capacitat de procés. Com dic, el problema és el mateix que amb Lucene: com que només tenim una instància d'aquesta plataforma, i no és precisament barat, no podem tenir una base de dades replicada en Exadata.

Podríem instal·lar algun SGBD capaç de funcionar entre múltiples data centers, però aquestes topologies no són senzilles i suposen una complicació important per l'equip d'infraestructures. Si tenim en compte el volum de dades i l'ús que n'hem de fer d'elles, optar per aquest tipus de solucions hauria estat matar mosquits a canyonades.

Finalment vam optar per una solució poc ortodoxa, i semblant a la del punt anterior. Persistim tot aquest model en un únic fitxer en JSON. Aquest fitxer està en una carpeta NFS i regularment se'n comprova la seva data de modificació per actualitzar-ne els canvis. Des de que arrenca l'aplicació, el model és mantingut en memòria, i per tant només es realitzen operacions d'IO quan s'escriuen canvis o quan es detecta de que el fitxer s'ha modificat (per un altre node).

Creiem que això és una bona solució si tenim en compte el següent:

- Es tracta d'un volum relativament petit d'informació. Actualment, el fitxer ocupa uns 160kB, però encara es podria reduïr considerablement. Conté molts espais, ja que per facilitar la lectura en cas de que ho vulgui fer un humà, no fem cap tipus de minimització. Hi ha alguna informació redundant que durant una primera fase de refactorització podrem treure.

- S'ha de tenir en compte que el model no va creixent gaire en el temps. No ens fa falta l'històric de les programacions passades, per tant només es guarda la planificació d'una setmana.

- El model no conté cicles. En cas d'haver-hi algun cicle s'hauria pogut solventar, però hauria complicat el procés de lectura i escriptura. Probablement hauríem hagut de crear un model intermedi sense cicles.

- No s'han de fer queries. Les operacions més freqüents dels repositoris serán les cerques d'àudios, que les fa Lucene. Consultar la informació dels programes i la graella també és molt freqüent, però com que el model es troba en memòria, no hi ha cap operació d'IO implicada.

- La freqüència d'escriptura és molt baixa. Està previst que l'Audioteca tingui una bona afluència, sí, però el model només es veu afectat en dos casos:
  - Quan l'usuari administrador entra al backoffice de l'Audioteca i modifica la graella o els programes. Probablement això passi menys de deu vegades al dia.
  - Quan la PortalPub avisa, a través de la seva API, de que des de Méthode s'ha modificat certa informació d'algun programa. Concretament, el títol, subtítol o descripció. Això segurament passarà amb encara molta menys freqüència.
- La serialització d'aquest model a un fitxer i el pas contrari només suposen un parell de línies de codi en cada cas. El mateix fet en JPA (l'ORM estàndard de Java) hauria implicat moltes més liníes. En el cas de volerho fer amb comandes SQL ja ni en parlem. A més, les conversions de tipus complexes (com per exemple, les dates en ISO 8601) són les mateixes que s'utilitzen per les API.

L'entitat que agrupa tota aquesta informació a persistir es diu State, i està formada per altres classes de la capa de model. La classe State, però, obeeix a una necessitat de la capa de repositori. Per tant és allà on ha d'estar i no al model comú.

```
@Data
public class State {
    private List<Program> programs = new ArrayList<>();
    private SortedMap<String, ScheduleTemplate> schedules = new TreeMap<>();
    private Map<DayOfWeek, ScheduleTemplate> plan = new HashMap<>();
}
```

I el fitxer state.json és de l'estil:

```
{
    "programs": [{
      "id": "el-mon",
      "methodeSectionId": "/programes/el-mon",
      "title": "El Mon a RAC1",
      ...
    }],
    "schedules": {
        "20ab5db3-9d19-4320-9a15-db971263db4d": {
            "id": "20ab5db3-9d19-4320-9a15-db971263db4d",
            "name": "Diaria",
            "programs": [...]
        },
        ...
    }
    ...
}
```

4.3.3 Servidor d'àudios amb publicitat

Aquest s'ha tractat de l'entrebanc més gran de tot el projecte. També es tracta de la part que, en la meva opinió, ha quedat més dèbil des del punt de vista tècnic. De tota manera, la solució actual s'ha considerat des del primer moment que seria provisional, amb una durada en producció d'uns mesos.

Per tal de servir àudios amb publicitat s'utilitzen les mateixes tècniques que pel cas dels vídeos. Bàsicament existeixen dos tipus de mecanismes, cada un amb els seus avantatges i inconvenients:

- Inserció en el client:
  - S'utilitza un player en JavaScript que s'encarrega de fer crides a servidors de publicitat, que li retornen la URL de la falca publicitària. El reproductor s'encarrega d'emetre-la (permetent que es salti o no) i després emet l'àudio que vol sentir l'usuari.
  - A més del cas descrit, el pre-roll (que mostra la publicitat abans de l'àudio desitjat), també permet post-roll (al final) o mid-roll, però aquest últim només està limitat a només mostrar anuncis cada certa freqüència
  - Encara que l'usuari vulgui escoltar un àudio, el player pot decidir mostrar-li un vídeo publicitari
  - Suposa molt poca càrrega al servidor i permet que s'utilitzin diferents tipus de cachés, com una CDN
  - Es tracta d'un blanc molt fàcil pels ad-blockers. Aquests es dediquen principalment a impedir o manipular les peticions a una llista negra de dominis de servidors de publicitat. Per tant no permetrà descarregar-se la falca però sí l'àudio que es serveix des d'un domini del client
- Inserció en el servidor:
  - La fusió entre l'àudio editorial i les falques es realitza en el servidor, produint un únic fitxer o stream. Aquesta tècnica es coneix amb el nom d'ad stitching
  - Permet els mateixos tipus d'insercions que quan es fa en client (pre-roll, post-roll i mid-roll). Però així com en el client l'in-roll només pot inserir un anunci cada certa freqüència, aquí existeix una opció més. És possible crear MP3 amb marques inaudibles que senyalen punts adequats per a inserir una publicitat. D'aquesta manera, s'evita tallar aleatòriament un àudio enmig d'una conversa o d'una cançó, per exemple. Aquest processament per trobar les marques és relativament fàcil en el servidor, però molt complicat fer-lo en JavaScript (quan a més aquest no processa l'stream, sinó que se n'encarrega un player en HTML5 o Flash). No sé de cap opció comercial capaç de fer-ho
  - No es pot inserir un vídeo a dins un fitxer d'àudio
  - Aquest processament evidentment té un cost pel servidor. A més, s'ha d'anar alerta si es decideix aplicar cachés. El servidor de publicitat pot decidir mostrar una falca o una altra depenent de molts factors (contingut de l'àudio, IP o dispositiu de l'usuari final, el moment del

dia, el recorregut de les diferents campanyes o inclús aleatòriament). Per tant, si dos usuaris demanen de sentir el mateix àudio, és molt probable que el fitxer que se'ls acaba servint no sigui idèntic.

- Els ad-blockers no poden fer res pel moment
- Existeixen solucions comercials que inclouen l'allotjament d'aquests servidors, i d'altres en que el client és qui s'encarrega d'instal·lar la plataforma en les seves màquines

Des de negoci, tant en la web antiga com la nova, sempre s'ha apostat per la segona opció. El servidor de publicitat que s'estava utilitzant fins ara es diu Adswizz. Està pensat per la inserció d'àudios tant en l'streaming en viu com en els fitxers d'àudio, i ja s'estava utilitzant per ambdues. Com que s'està content amb el resultat i no entra dins l'abast del projecte ni les responsabilitats de l'equip tècnic, amb el nou desenvolupament s'ha mantingut el mateix proveïdor.

El que sí es volia fer era actualitzar la forma en que s'utilitza per servir àudios sota demanda. La forma recomanada per Adswizz és instal·lar la seva plataforma en el client. Aquesta és l'encarregada de fusionar en temps real els fitxers d'àudio.

Quan es va fer la web antiga, però, per certs motius es va decidir fer-ho d'una forma alternativa: utilitzant una crida VAST (Video Ad Serving Template). Es tracta d'una especificació estàndard molt estesa per comunicacions relatives a anuncis de vídeo (com el seu nom indica) i àudio. Per tant, el procés pel que un usuari acabava rebent un àudio a la web antiga era el següent:

- L'usuari feia una petició al servidor de RAC1 per escoltar un àudio
- A partir del nom del fitxer demanat, el servidor deduïa de quin programa i secció es tractava. A partir d'un mapping d'ids d'Adswizz segons la secció, feia una crida VAST al servidor d'Adswizz passant aquest id. Al servidor Publipress havia configurat una sèrie de campanyes per a cada secció/id
- El servidor responia amb un XML contenint la URL d'una falca (o no). La mateixa resposta també contenia una altra URL per notificar al servidor d'Adswizz de que efectivament s'havia servit aquella impressió
- El servidor concatenava els dos MP3 i els servia finalment a l'usuari
- Es feia caché dels fitxers resultants, però només es podia servir quan la crida a Adswizz retornava la mateixa falca per un mateix àudio

Aquesta solució, que a primera vista no té molt mala pinta, tenia alguns problemes:

- El merge de dos MP3 no es tant senzill com un pot imaginar. Com que sembla que en el seu moment no es va trobar cap llibreria en PHP que ho permetés, es va decidir fer una concatenació directa dels dos fitxers, fent una crida de sistema.

- Això, sorprenentment, funciona. Però obliga a que els MP3 de les falques i dels àudios siguin molt semblants: mateix bitrate (i per tant, constant), mono/estèreo, sense tags ID3, etcètera. Per tant, totes les falques (creades pels anunciants o agències) s'havien d'editar, s'havia de tenir cura de que tots els àudios que es pujaven al servidor fossin correctes, etcètera.

La plataforma d'Adswizz evidentment gestiona tot això, a més de proporcionar algunes funcionalitats molt interessants:

- Enlloc de parsejar el nom per obtenir l'id publicitari busca aquesta informació en els tags ID3
- Permet fer targeting, a més de per l'id anterior i l'hora, per l'IP o el dispositiu
- No es necessari fer crides a Adswizz per a cada petició, ja que aquesta plataforma va sincronitzant-se amb les regles del servidor
- Fa comprovacions regulars a la font dels MP3 per veure si han canviat
- Implementa HLS. Es tracta d'un format cada vegada més popular pels dispositius mòbils. Bàsicament es tracta d'un format d'streaming que permet modificar el bitrate depenent de la qualitat de la línia, sense que l'usuari en noti interrupcions
- S'utilitza la mateixa plataforma per l'on demand i l'streaming en directe
- Permet diferents arquitectures, com mestre-esclaus, load balancings, etc

Sembla clar que instal·lar el software d'Adswizz era la millor opció. El problema va ser quan, durant el juliol vaig contactar amb ells per tal de posar-ho en marxa. Ja des de la primera videoconferència amb el comercial, en explicar-li la urgència, no va semblar molt convençut. Després d'una altra reunió més tècnica em van explicar que no era possible instal·lar el seu software abans d'octubre o novembre. No es tracta d'un software pensat perquè l'instal·li i configuri el client, sinó que ells ajuden a fer un estudi per determinar quins han de ser els paràmetres idonis. Tenint en compte les dates (estiu) i la càrrega de feina que ja tenien en aquell moment, no hi havia res a fer.

Ens vam plantejar si utilitzar el mateix desenvolupament en PHP, però tampoc era trivial. Obtenir la secció parsejant el nom del fitxer implicava haver de seguir pujant els fitxers a aquestes màquines amb el nom antic. Com que el mapping entre secció i id publicitari estava en un fitxer en els servidors, no seria possible des del backoffice de l'Audioteca crear seccions amb nous ids publicitaris sense que l'equip tècnic intervingués (actualitzant el fitxer de mappings).

Existia la possibilitat de modificar el PHP tenint en compte tot això, però no era trivial. Finalment vam decidir implementar un esquema molt semblant al que hi havia, però des de l'Audioteca:

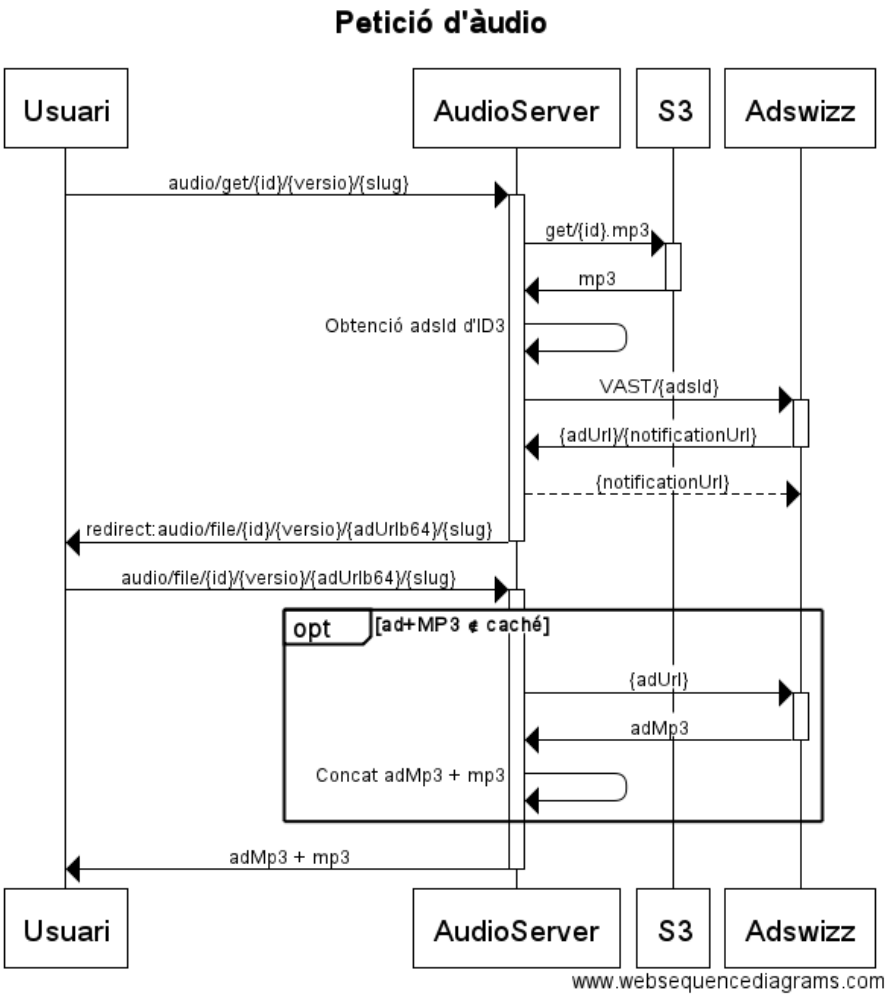

Figura 14: Servei d'àudios amb publicitat

- En realitat, el que hem fet són dos modes d'arranc. Unes variables d'entorn indiquen si ha d'actuar en mode AUDIOTECA o AUDIO_SERVER. Segons el cas, s'inicialitzaran uns beans o uns altres i aplicaran unes propietats lleugerament diferents

- Les audioteques estaran a les màquines de Grupo Godó, mentre que, per donar l'ample de banda necessari (i més barat), els audio servers estaran allotjats en servidors externs

- La petició que fa l'usuari conté un "slug"per deixar unes URL més boniques. També conté una versió, per evitar que retornem de la caché un àudio amb una versió més nova.

- Hem de fer un redirect a una URL que identifiqui l'àudio amb la publicitat. Això és degut a que habitualment, pels fitxers grans, els exploradors es baixen els fitxers a trams. Si a cada petició fessim una nova crida a Adswizz tindríem dos problemes:

  - Es comptarien moltes impressions falses
  - Si cada vegada s'inserís una publicitat diferent, l'usuari acabaria rebent un MP3 corrupte

- Com que aquesta segona petició pot anar a un node AUDIO_SERVER diferent, i entre ells no tenen comunicació, necessitem passar la URL sencera (en Base64) de la URL de la falca

4.3.4 Migració

El servei antic d'"a la carta", no tenia base de dades ni cap altra forma de persistir dades. Els MP3 tampoc tenien informació guardada, com ara. Per tant la única informació que teníem era el nom del fitxer o la carpeta on es troba.

Els diferents patrons eren de l'estil:

- Talls horaris: /mp3/0706 18h (Dimecres 06-07-16) VERSIO RAC1.mp3

- Seccions: /el_mon_a_rac1/0706 10h (Dimecres 06-07-16) Entrevista (Jordi Moix).mp3

- Píndoles: /audios_elmon/100701_Joan_Manuel_Tresserras.mp3

Hem creat patrons pels tres tipus d'àudios i em mapejat cada carpeta al programa que correspon. En arrencar el procés ens vam trobar amb alguns problemes:

- Noms de programes que no existien. La majoria es tractava d'especials (programes de Nadal i simil·lars)

- Seccions inexistents, amb poques ocurrències

- Diferents patrons al nom de les píndoles

Estem esperant la resposta de negoci perquè decideixi que fer en aquests casos. Ens sembla una mala idea crear programes que només tenen un parell d'emissions. Tots aquests programes després sortiran al llistat de programes per cercar, lo qual no agrada. Si no han de sortir en aquest llistat tampoc té cap sentit migrar-los, ja que no seran accessibles. Creiem que haurem de fer dues coses:

- Assimil·lar el programa original a l'especial: *La competència de Nadal* = *La competència*
- Crear un program anomenat *Altres* que contingui els especials que no corresponen a cap programa

Un altre problema que hem tingut en compte és la generació dels ids dels àudios migrats. A la web antiga es troben notícies que contenen links de l'estil "http://racalacarta.com/audio/audios_elmon/160711_Neus_Munte.mp3". Quan l'equip web migra la nova notícia, ha de col·locar a dins l'id de la píndola migrada, que ha de coincidir amb l'id de l'àudio migrat. Una opció hauria estat que un dels dos fes tota la migració, creant ids aleatoris, i després passi el llistat de parells url-id a l'altre equip.

Tenint en compte el calendari, no ens podíem permetre que un hagués d'esperar a l'altre. Vam decidir que els dos generaríem l'id a partir del nom del fitxer. Aquest és el mètode que fa la transformació:

```
1     public static Pair<String, UUID> getIdByName(File file) {
2         val id = UUID.nameUUIDFromBytes(file.getName().getBytes(StandardCharsets.UTF_8));
3         return Pair.makePair(file.getName(), id);
4     }
```

Pel moment de la sortida s'han migrat la gran majoria dels àudios que es trobaven al servidor antic, uns 40000. El procés de migració no ha estat ràpid. Com que tampoc teníem excessiva presa hem utilitzat un ordinador no molt potent. Després hem pujat en batch tots aquests fitxers, que sumats sumen uns 500GB, a S3 amb una eina que proporciona Amazon. Tota elprocés de migració ha durat més de tres dies, en total.

4.4 Instal·lació i manteniment

La instal·lació de l'aplicació és molt senzilla. Des del servidor d'integració continua, Jenkins, es crea un fitxer WAR i es publica a l'Artifactory. Quan es vol fer un desplegament se li passa l'enllaç d'aquest fitxer a l'equip d'operacions. Reiniciant el Tomcat és suficient.

En el cas de que les carpetes necessàries (locals i compartides) no estiguin creades, es generen durant la inicialització.

Durant el primer deploy, però hem hagut de tenir en compte un altre factor. Per a fer la migració, ho hem fet des de màquines independents, bàsicament per processar els fitxers MP3, pujar-los a S3 i generar els .json. Recordem que a l'Audioteca hi ha un procés que va monitoritzant una carpeta i introdueix els

fitxers a Lucene quan detecta els canvis. Però 40000 canvis de cop són massa. Bé, en realitat tampoc sabem si són massa. Però quan hem vist que es tardava més de deu minuts hem decidit parar i anar copiant els fitxers a intervals més petits.

El manteniment de l'aplicació, a nivell d'operacions, no implica més feina que qualsevol altra aplicació. Hi ha una eina de monitorització, Zabbix, que cada cert temps fa una petició /check a l'Audioteca. Quan aquesta no retorna s'encén una alerta a la pantalla dels operadors.

La configuració del sistema es realitza editant un fitxer clau/valor: application.properties. Actualment té 125 propietats. Per a que els canvis siguin visibles s'ha de reiniciar l'aplicació. El fitxer conté alguns comentaris explicant per a què serveix la propietat i el seu format.

També es pot modificar un fitxer logging.properties que especifica el nivell de log per cada classe que els emet.

De totes formes, si els operadors necessiten realitzar cap modificació la poden fer en voler, però els hi demanem que ens avisin per també nosaltres modificar-lo en l'origen. D'altra forma, els seus canvis es sobreescriurien quan despleguessim una nova versió.

Exemple de propietats:

```
1
2  # Hora en que comencen les plantilles de la programacio. En ISO 8601
3  schedule.start=06:00
4
5  # Durada dels fitxers temporals de la carpeta incoming fins que son esborrats. En ISO 8601
6  incoming.duration=P3d
7
8  # Carpeta local per a cada node
9  path.base.local=/mnt/tomcat/audioteca/rac1
10 LOCAL.path.base.local=/tmp/audioteca/rac1
11
12 # Carpeta compartida entre tots els nodes per NTFS
13 path.base.shared=/mnt/apps/audioteca/rac1
14 LOCAL.path.base.shared=/tmp/audioteca/rac1
15
16 # Lucene
17 path.index=${path.base.local}/index
18
19
20 # Credencials S3
21 s3.bucket=ract-audioteca
22 s3.user=audioteca-tes
23 s3.key.secret=xxx
24 s3.key.access=xxx
```

5 Planificació i valoració econòmica

5.1 Calendari

Quan es va fer la presa de requisits i la planificació, es van decidir dues fites:

La primera pel 15 de juliol. Es tracta del moment en que uns quants usuaris, redactors de RAC 1, havien de començar a rebre la formació en el client de Méthode. En aquest moment havien de poder pujar àudios a l'Audioteca i veure el resultat a dins la web de RAC 1, a l'entorn de test.

L'altra era la del 6 de setembre, la sortida en producció de la nova web.

Tenint en compte això es va fer un calendari pensat per a tres recursos. Un d'ells estava especialitzat en ZK, i per tant havia de fer el backoffice. Un altre gestionaria la integració amb Méthode. Qui havia de fer gran part de les tasques era jo.

Segons el calendari, molt optimista, l'11 d'agost s'arribava al final del desenvolupament. A l'informe previ del PFC ja advertia d'algunes coses que s'havien de tenir en compte: no es podia suposar una dedicació plena per la meva part, ja que hi ha altres tasques que requereixen la meva participació, i a més ens trobaríem ja en perillós moment en que tothom està de vacances.

Com acostuma a passar, aquestes advertències es van quedar curtes. Diversos problemes més urgents que van succeir entre meitat de juliol i principis d'agost, va fer que pràcticament no pogués avançar.

Quan els usuaris van començar la formació, ja es podia veure el backoffice per gestionar programes i pujar àudios, però era tot atrezzo. Els àudios no anaven enlloc, no s'indexava el contingut a Lucene i no hi havia cap pàgina començada per mostrar un player.

Quan AdsWizz ens va confirmar que no podria instal·lar la plataforma per ara, ja vam veure que el calendari començaria a sofrir tensions. Fer els audioServer ens ha portat una bona quantitat de feina.

Inicialment també havíem pensat que l'agent de gravació no faria falta tenir-lo per la sortida, i que podríem seguir uns dies amb l'agent antic (que gravava a l'entorn antic) i algun procés que copiés contingut d'un a l'altre. Tenir això, a la vegada que també volem copiar els continguts de l'Audioteca al sistema antic per mantenir compatibilitat en l'app hauria estat una mica estrambòtic.

També hi va haver un parell de canvis que van reduir la nostra càrrega de feina. En un inici, es volia que el player *estàtic* tingués una llista de reproducció, a més de mostrar els essencials. Per motius de disseny, que no acabava de convèncer, això es va simplificar, deixant el player que tenim ara.

També s'havia pensat en llistes d'àudios. Es tractaria d'un grup d'àudios que generarien els redactors i que es podrien enllaçar, compartir, etc. Amb el temps es va perdre interès per aquesta funcionalitat.

Amb tot això, però, ens vam trobar en que a meitat/finals d'agost estàvem començant realment l'aplicació. Vam decidir canviar una mica el calendari. O millor dit, a no seguir-lo al peu de la lletra. L'objectiu era anar començant totes les tasques possibles, sense acabar-ne cap, per dos motius: poder trobar-nos tant aviat com fos possible amb problemes, i que més mans es poguessin sumar al

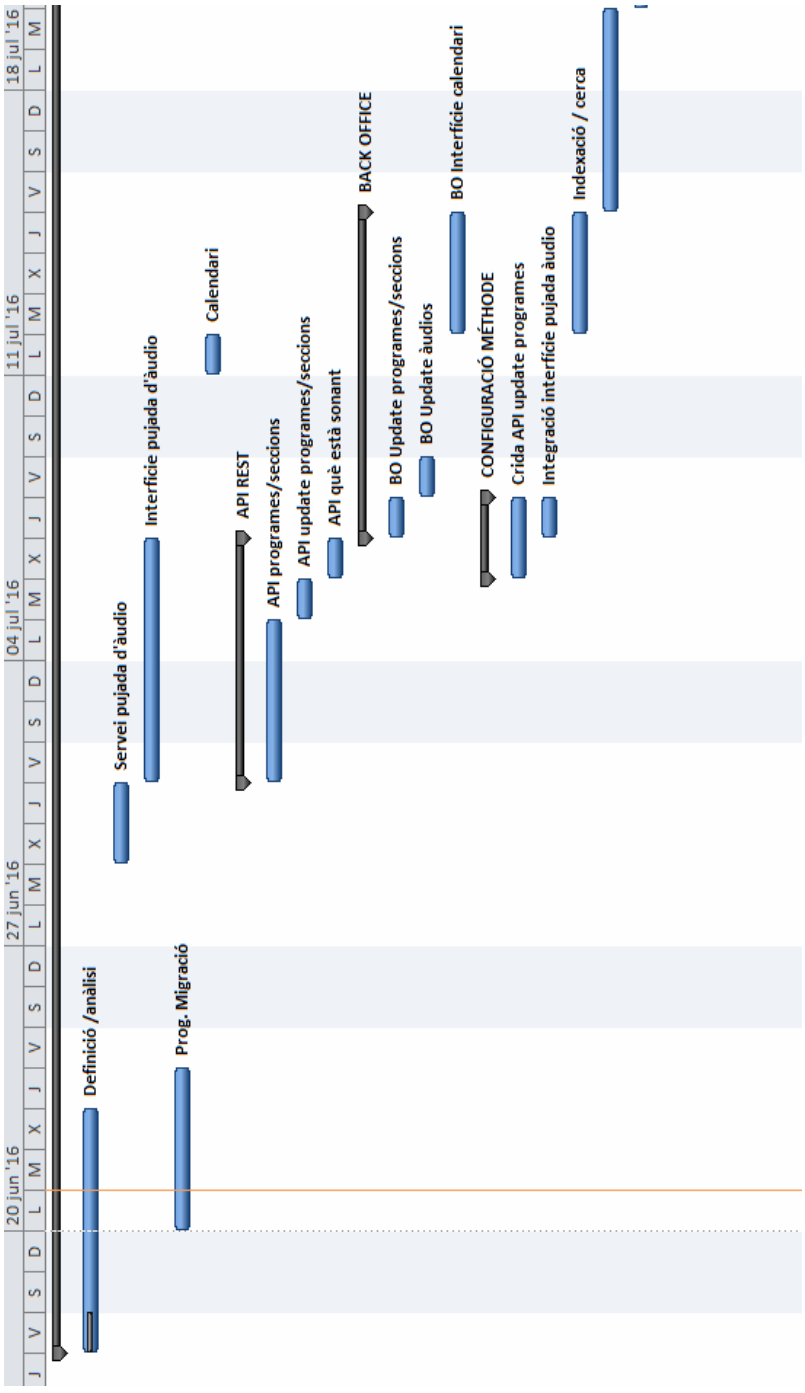

Figura 15: Calendari de desenvolupament proposat (1/2)

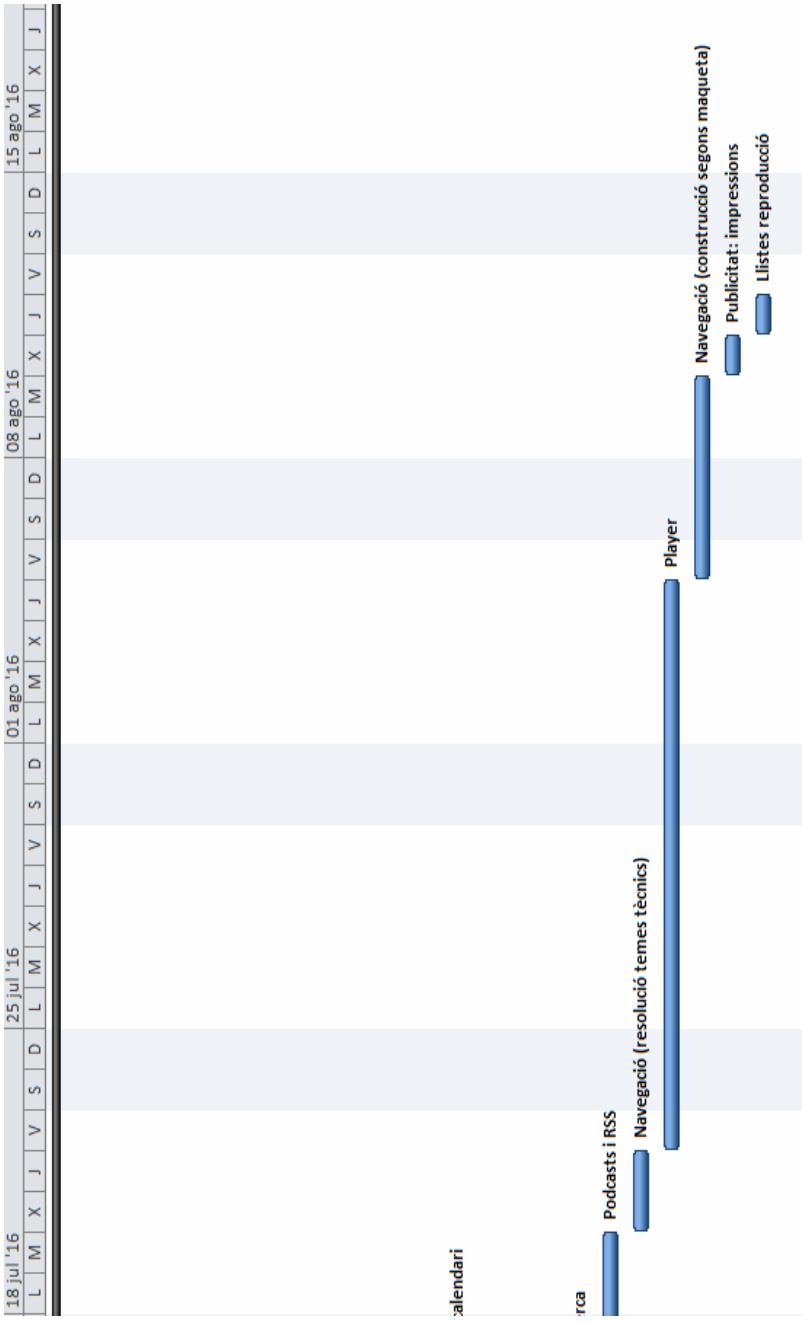

Figura 16: Calendari de desenvolupament proposat (2/2)

desenvolupament.

D'aquesta forma, enlloc de les tres persones que havíem de desenvolupar l'Aplicació, hem estat quatre. Una s'ha dedicat a fer els audioServer i l'agent i el que havia de fer backoffice s'ha dedicat a fer pàgines de l'Audioteca i a la migració. Jornades llargues i relegar la redacció d'aquesta memòria al final del projecte i a hores lliures, han permès acabar satisfactòriament.

Finalment, la data de la sortida de la web s'ha vist postergada al dia 26 de setembre, vint dies de diferència. La causa, però, no ha estat l'Audioteca. En honor a la veritat, he de dir que el dia 6 tampoc estava acabada.

En definitiva, en aquests moments consider-ho que només queden petits detalls, i que de fet sense solventar-los es podrà sortir d'aquí a quatre dies sense problemes.

5.2 Cost

Els costos associats als tres desenvolupadors en que s'havia pensat sumen 16570€. Calculo que el retard hauria implicat un desviament aproximat del 30%, que deixa el total en uns 21500€.

Tasca	Java	Frontend	Méthode
Definició/anàlisi	4	0	0
Servei pujada àudio	2	0	0
Interfície pujada	0	4	0
Migració	4	0	0
Servei calendari	1	0	0
API REST	4	0	0
Backoffice - programes	1	0	0
Backoffice - àudios	1	0	0
Backoffice - calendari	0	3	0
Méthode - programes	0	0	2
Méthode - pujada	0	0	1
Índex	3	0	0
Podcast	3	0	0
Navegació - temes tècnics	2	0	0
Navegació - pàgines	0	3	0
Player	10	0	0
Publicitat - impressions	1	0	0
Llistes de reproducció	1	0	0
Suma	37	10	3
Cost jornada	360€	250€	250€
Cost	13320€	2500€	750€
TOTAL			**16570€**

6 Conclusions

6.1 Millores proposades

Hi ha una sèrie de millores que segurament haurem de fer en breu

6.1.1 AdsWizz

Tal com he dit, encara que ja hem fet la feina per suplir la seva plataforma, instal·lar-la encara ens donaria molts avantatges: HLS, possibilitat de traficar publicitat per IP i dispositiu, poder fusionar fitxers MP3 amb diferents paràmetres, etc.

Segons el que van explicar, tampoc és una feina de dos dies. S'han de fer uns càlculs tenint en compte els número de fitxers que es baixen, quina mida i amb quina freqüència. Després, s'ha de decidir la topologia dels nodes i s'ha de fer una feina amb el departament d'infraestructures per configurar els paràmetres.

Tenint en compte les dates en que ja estem, i que fa setmanes que no parlem amb ells, calculo que ho podrem tenir fet cap a finals d'any.

6.1.2 Migració

Hem migrat tots els àudios que hi havia al servidor antic, que són aproximadament els de l'últim any. Sabem que, repartits en diferents màquines, hi ha fitxers de fa més temps. Quan ens els passin, haurem d'executar el procés de migració sobre ells.

Probablement ens trobarem alguns programes i moltes seccions que hauran de ser creats o ignorats o integrats a altres programes. Es tracta de decisions de negoci.

6.1.3 Player

Molt probablement el player es voldrà canviar:

- Una llista de reproducció que es quedi guardada en el *localStorage* de JavaScript serà segurament la primera demanda
- Vull intentar que quan s'està escoltant el player estàtic (àudios de la llista o el directe en streaming) i a una altra pestanya es vol sentir la píndola d'una notícia, per exemple, el player estàtic es pari i es torni a arrencar en acabar de sentir la píndola. Pel que he llegit hauria de ser possible la comunicació cross tab usant també l'objecte *localStorage*

6.1.4 Backup

Crec que també seria interessant una operació d'utilitat que pugui realitzar un backup de tot el contingut de l'índex. Podria un zip amb fitxers semblants als que ara es guarden a la carpeta d'incoming. Entre altres coses, ens podria servir per actualitzar més fàcilment els entorns de preproducció o local. D'aquesta forma podem fer proves amb dades més realistes.

6.2 Valoració del projecte

Feia temps que no em podia dedicar a un mateix projecte al 90%. En general em dedico més a estar a la fase inicial d'alguns projectes, donar suport en certes tasques, o resoldre problemes que cremen. La veritat és que estar a un projecte d'inici a fi és una cosa que crec que em feia falta i m'ha agradat.

La part negativa, sens dubte, ha estat un calendari massa apretat. Tot i que ja fa dies que considero el projecte acabat fins el dia de la sortida en producció, fa unes setmanes no tenia clar que arribessim així de bé.

Per altre costat, tot i haver utilitzat Java 8, que m'apeteixia bastant, no he utilitzat cap tecnologia que fos innovadora.

Finalment, aquesta memòria, que s'ha realitzat en gran part durant els últims dies, ha estat la part a la que menys acostumat estic i la que se m'ha fet més dura. No és que hagi estat un infern, però ha deixat en evidència que ja no estic gens al dia en aquests tipus de feines. Això sí, aprendre LaTeX, que no l'havia fet servir mai, se m'ha fet molt interessant.

7 Annex. Imatges de l'aplicació

Figura 17: Home de la nova web. L'audioteca mostra el que està sonant ara i el llistat de players

Figura 18: Detall d'un player en una notícia

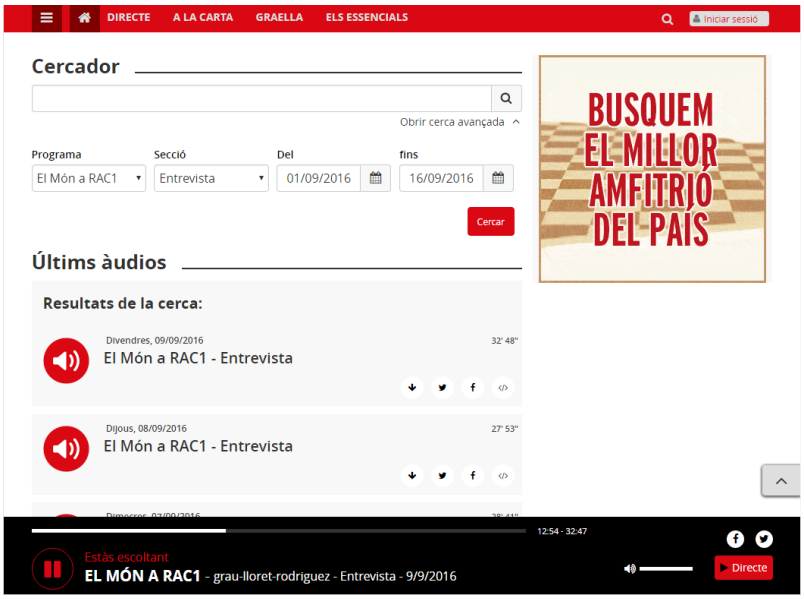

Figura 19: A la carta. Resultats de cerca i playes estàtic

Figura 20: Detall de graella de programes

Figura 21: Notícia per a compartir a Facebook i Twitter

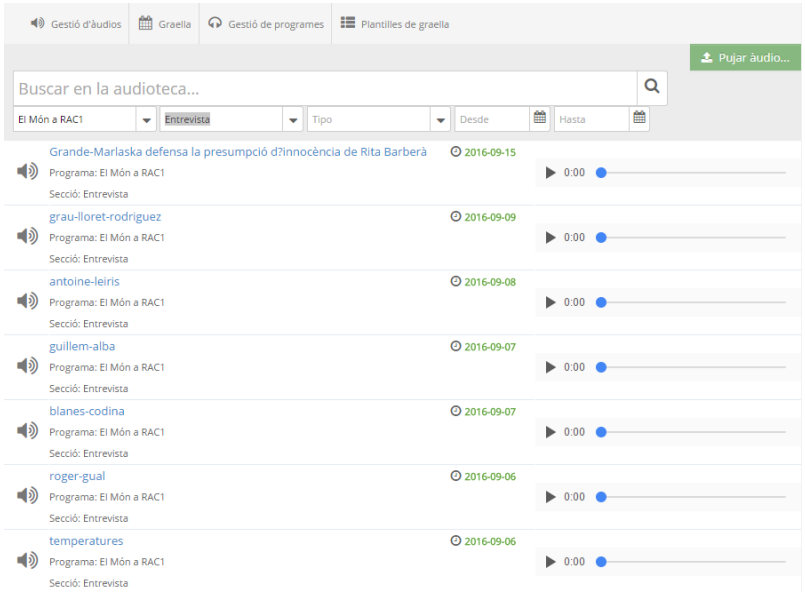

Figura 22: Backoffice. Gestió d'àudios

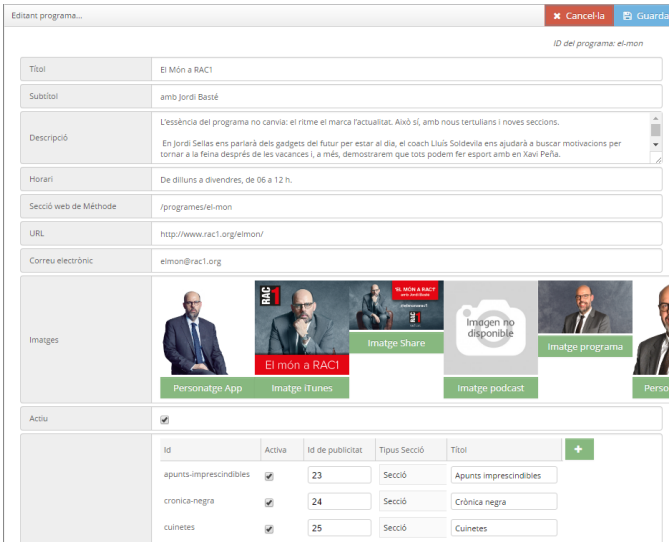

Figura 23: Backoffice. Gestió de programes

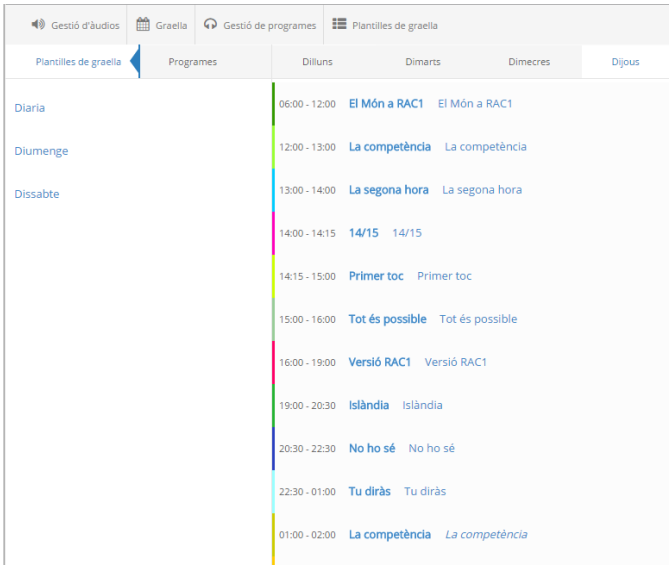

Figura 24: Backoffice. Gestió de graella